ESL-TR-91-22
Volume X

THE POST-DAM SYSTEM VOLUME X - INTEGRATED DATABASE MANAGEMENT AND PROJECT SCHEDULING SYSTEM FOR POSTATTACK DAMAGE ASSESSMENT

M.D. SMITH, M.C. SAWYER, R.H. SUES

APPLIED RESEARCH ASSOCIATES, INC,
POST OFFICE BOX 40128
TYNDALL AFB FL 32403

OCTOBER 1992

FINAL REPORT

JUNE 1991 - APRIL 1992

APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED

ENGINEERING RESEARCH DIVISION
Air Force Civil Engineering Support Agency
Civil Engineering Laboratory
Tyndall Air Force Base, Florida 32403

NOTICE

PLEASE DO NOT REQUEST COPIES OF THIS REPORT FROM HQ AFCESA/RA (AIR FORCE CIVIL ENGINEERING SUPPORT AGENCY). ADDITIONAL COPIES MAY BE PURCHASED FROM:

NATIONAL TECHNICAL INFORMATION SERVICE
5285 PORT ROYAL ROAD
SPRINGFIELD, VIRGINIA 22161

FEDERAL GOVERNMENT AGENCIES AND THEIR CONTRACTORS REGISTERED WITH DEFENSE TECHNICAL INFORMATION CENTER SHOULD DIRECT REQUESTS FOR COPIES OF THIS REPORT TO:

DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22314

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFEID

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release.
Distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
5691

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESL-TR-91-22 Volume X

6a. NAME OF PERFORMING ORGANIZATION
Applied Research Associates, Inc.

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
HQ AFCESA/RACS
Tyndall AFB FL 32403

6c. ADDRESS (City, State, and ZIP Code)
Southeast Division
6404 Falls of Neuse Road, Suite 200
Raleigh, North Carolina 27615

7b. ADDRESS (City, State, and ZIP Code)

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Air Force Civil Engineering Support Agency

8b. OFFICE SYMBOL (If applicable)
RACS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)
Air Base Survivability Branch, Engineering Research Division
Air Force Civil Engineering Support Agency, HQ AFCESA/RACS
Tyndall Air Force Base, FL 32403-6001

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO

11. Title (Include Security Classification)
Integrated Database Management and Project Scheduling System for Postattack Damage Assessment

12. PERSONAL AUTHOR(S)
Smith, M. D., Sawyer, M. C., and Sues, R. H.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 6/1/91 TO 4/15/92	October 1992	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Damage Assessment, ABO, POSTDAM, Expert System, Resource Manager/Repair Scheduler (RMRS)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In a postattack environment, field information on mission-critical facility damage is collected and analyzed to determine structural integrity and usability. From this analysis, a repair schedule is developed. This process is time-consuming when unaided by a computerized system. Consequently, the POST-DAM system was developed to determine repair strategies using an expert system; keep track of materials and equipment using a relational database management system; and schedule repairs based on manpower and equipment availability using a project management system. This report addresses the combination of the relational database management system and the project management system of the original POST-DAM architecture into one integrated RMRS system.

The RMRS prototype software demonstrates an architecture that is able to quickly and efficiently manage Air Base resources and schedule expedient repairs for mission critical facilities. For repairs with no resource conflicts, the RMRS system operates with minimal user intervention. When conflicts are present, resolution is possible by allowing the user direct access to the RMRS data files and complete control of the resource allocation and scheduling processes. User interaction is completely menu-driven, with context-sensitive help to minimize training requirements and maximize system integrity.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
[X] UNCLASSIFIED / UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Including Area Code)	22c. OFFICE SYMBOL
MAJ. JACOB GHERIANI		HQ AFCES/RACS

DD FORM 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

(The reverse of this page is blank.)

EXECUTIVE SUMMARY

A. OBJECTIVE

This report describes the software developed by Applied Research Associates, Inc. for the host computer portion of the airbase facility postattack damage assessment (POST-DAM) system. The objective of this research was to develop an integrated prototype software system to manage repair resources and schedule repairs based on resource availability and facility priority. This software system, entitled Resource Manger/Repair Scheduler (RMRS), was design to be user friendly and require minimal user interaction.

B. BACKGROUND

In a postattack environment, field information on mission-critical facility damage is collected and analyzed to determine structural integrity and usability. From this analysis, a repair schedule is developed. This process is time-consuming when unaided by a computerized system. Consequently, the POST-DAM system was developed to determine repair strategies using an expert system; keep track of materials and equipment using a relational database management system; and schedule repairs based on manpower and equipment availability using a project management system. This report addresses the combination of the relational database management system and the project management system of the original POST-DAM architecture into one integrated RMRS system. The original POST-DAM work is documented in a nine-volume set. Volume I describes software and hardware used with the prototype POST-DAM system, and recommends software and hardware for full-scale development. Volumes II through VIII are software user's manuals that describe how to install and use the prototype software with the POST-DAM system. Volume IX is a field manual that contains diagrams of structures that are used with the POST-DAM system to locate damaged elements.

C. CONCLUSIONS

The RMRS prototype software demonstrated an architecture that was able to quickly and efficiently manage Airbase resources and schedule expedient repairs for mission critical facilities. For repairs with no resource conflicts, the RMRS system could operate with minimal user intervention. When conflicts were present, resolution was possible by allowing the user direct access to the RMRS data files and complete control of the resource allocation and scheduling processes.

D. RECOMMENDATIONS

For full-scale development of the RMRS software, the following issues should be addressed:

1. Include a communications module to allow RMRS to receive PDES data files in the background while continuing to work uninterrupted.

2. Incorporate Expert System capabilities to provide RMRS the capability to suggest resource conflict solutions and changes in repair strategies based on current resource supplied and equipment usage.

3. Develop an optimized scheduler to generate schedules based on facility priority and time and resource optimization.

PREFACE

This report was prepared by Applied Research Associates, Inc. (ARA), 6404 Falls of Neuse Road, Suite 200, Raleigh, NC 27615, under Contract F08635-88-C-0067, for the Air Force Civil Engineering Support Agency, Tyndall Air Force Base, Florida.

This report (Volume X) summarizes work completed between 1 June 1991 and 31 December 1991. Lt. James Underwood (USN) and Maj. Jacob Gheriani (IAF) were the HQ AFCESA/RACS Project Officers.

This report has been reviewed by the Public Affairs Office, and is releasable to the National Technical Information Service (NTIS). At NTIS it will be available to the public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Dan Shenbach, Maj., IAF
Project Officer

Felix Uhlik, Lt. Col., USAF
Chief, Engineering Research Division

William S. Strickland
Chief, Airbase Survivability Branch

Frank P. Gallagher, III, Col., USAF
Director, Civil Engineering Laboratory

(The reverse of this page is blank.)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION I

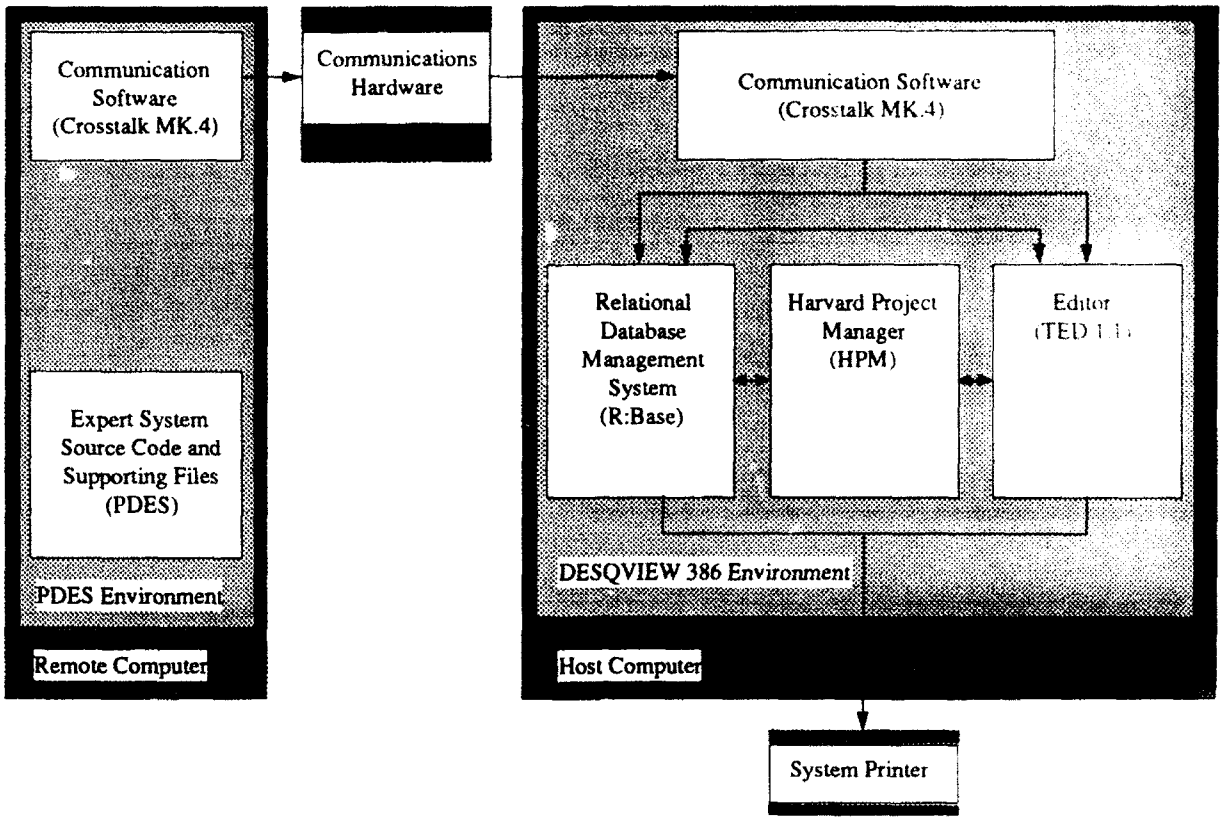
INTRODUCTION

A. OBJECTIVE

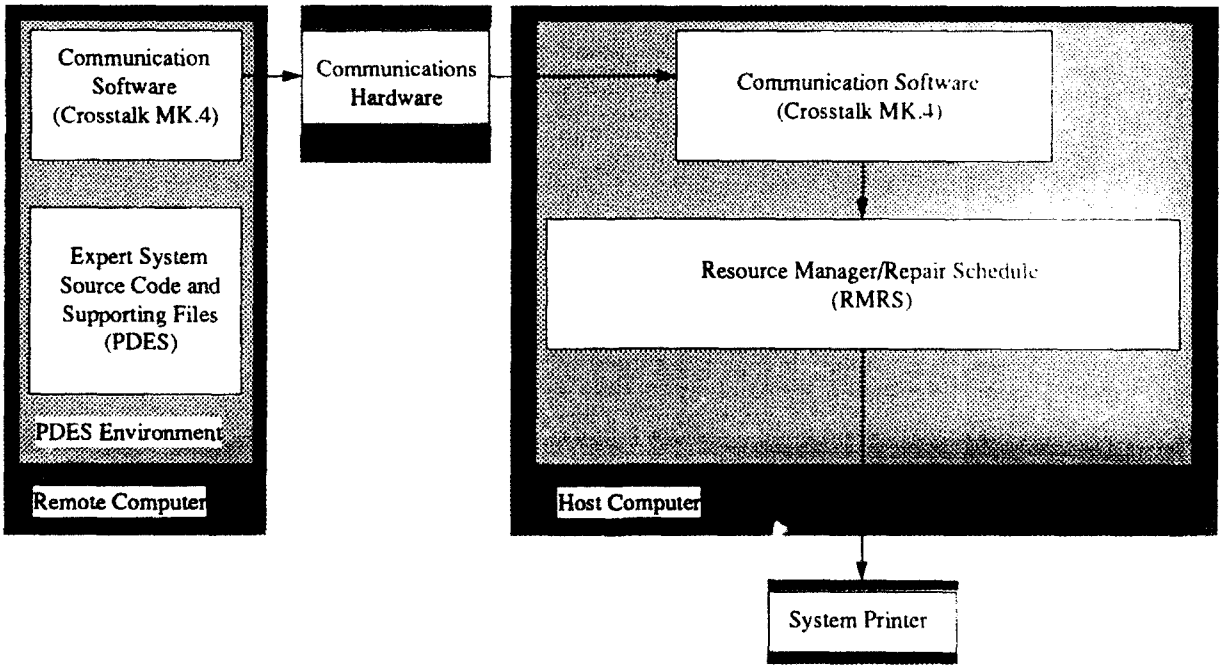
Our objective is to develop a personal computer-based software system to aid base engineers in postattack facility damage assessment (POST-DAM) of mission-critical facilities. This system is to integrate a database manager to track airbase material and equipment resources and a facility priority-based repair scheduler. The result of this effort is the Resource Manager/Repair Scheduler (RMRS) host system, which works in conjunction with the previously developed POST-DAM expert System (PDES).

B. BACKGROUND AND SCOPE

The RMRS host module was developed in response to limitations identified during testing of the early POST-DAM System prototype. In its original form, the host system portion of POST-DAM consisted of four separate commercial off-the-shelf (COTS) software packages and required more user interaction and training than was desired. Thus, a custom application was developed that combined the resource supply management function of the relational database manager and the repair scheduling function of the project scheduler. The RMRS module was designed for optimal host system automation, requiring user intervention only under special conditions, while allowing the host system operator complete control over resource usage and repair scheduling. Figure 1 shows the difference in the POST-DAM architecture with RMRS.



a. Early POST-DAM System Architecture.



b. New POST-DAM System Architecture.

Figure 1. Early and New POST-DAM Architectures.

SECTION II

SYSTEM TECHNICAL DESCRIPTION

A. GENERAL

RMRS is a tool to aid Damage Control Center (DCC) personnel in efficiently allocating resources and schedule expedient repairs to mission-critical Air Force base facilities following an enemy attack. The RMRS module operates as part of the Postattack Damage Assessment System (POST-DAM).

The RMRS module was developed using the C Language and, when possible, third party libraries for such functions as the low-level database file management, user interface, etc. It is menu driven with context sensitive help and mouse support. A major design criterion was that the system operate with minimal user interaction. To this end, the RMRS module can generate repair schedules and perform the following functions with as few as three commands:

Command: PROCESS | AUTO MODE [1]

- Retrieves POST-DAM EXPERT SYSTEM (PDES) from host hard disk storage.
- Creates entry in RMRS database .or each repair.
- Compares equipment requirements for repair with available resources.
- If the equipment resources are available, RMRS will continue; otherwise RMRS will let the user know that this repair is not possible.
- Compares material requirements for repair with available resources.
- If the resources are available, RMRS will allocate the resources to the repair; otherwise RMRS will enter an interactive mode to allow the user to resolve the resource conflict.

Command: SCHEDULE | FULL SCHEDULE

- Makes list of all repairs to be scheduled in facility priority order.
- For each repair, finds the earliest time slot available for all equipment resources required and allocates this time slot to the repair.

Command: GEN REPORT | REPAIR SCHEDULE

- Based on results of the schedule process, outputs schedule information in desired format.

[1]This terminology **PROCESS | AUTO MODE** refers to the *Process* menu and the *Auto Mode* choice.

B. DATABASE SCHEMA

The RMRS system is organized around seven data files, or databases, that maintain the required information to track and schedule materials, equipment, and repairs.

The seven databases are:

1. FACPRT.DBF - Airbase facility priority listing.
2. EQPSUP.DBF - Airbase equipment supply listing.
3. MATSUP.DBF - Airbase materials supply listing.
4. REPAIR.DBF - status records for each individual repair.
5. EQPREQ.DBF - records for each equipment requirement associated with repairs.
6. MATREQ.DBF - records for each material requirement associated with repairs.
7. REPINFO.DBF - misc. info records for each individual repair.

The first three databases are populated from existing Airbase-specific data in the system initialization phase. The remaining four databases are populated with the data that is received from the PDES. Each processed repair will have at least one related record in the REPAIR.DBF, EQPREQ.DBF, MATREQ.DBF, and REPINFO.DBF databases (excluding the special case where a repair does not require any materials, in which case, no material record would exist in MATREQ.DBF for that repair). The following sections describe the format of and the information contained in each of the RMRS databases.

1. FACTPRT.DBF

The database file FACPRT.DBF correlates the Airbase facility designations to their corresponding mission-critical priority rating.

FIELD#	FIELD NAME	FIELD DESC
1	FACNUM	Aibase facility number designation
2	PRIORITY	Airbase-assigned mission critical facility priority

2. EQPSUP.DBF

EQPSUP.DBF represents the current Airbase equipment supply. Only large equipment, such as bulldozers, shotcrete machines, etc., is included in this database. Smaller equipment, such as hammers, screwdrivers, etc., is considered to be standard equipment carried by all damage repair teams. Additionally, damage repair teams, containing approximately five members, are included in this database. The record format consists of a unique equipment identification number and a textual description of the equipment piece.

FIELD#	FIELD NAME	FIELD DESC
1	EQPID	RMRS system-assigned equipment id number
2	EQPDESC	Textual equipment description

3. MATSUP.DBF

MATSUP.DBF represents the current unallocated Airbase material supply. The record format consists of a unique material identification number, a textual description of the material, the total quantity of the material available, and a textual "units" description (e.g., ea, gal).

FIELD#	FIELD NAME	FIELD DESC
1	MATID	RMRS system-assigned material id number
2	MATDESC	Textual material description
3	QTY	Quantity of unallocated material units resident in Airbase supply
4	UNIT	Textual "units" description (e.g., ea, gal)

4. REPAIR.DBF

The REPAIR.DBF database consists of records (one per repair) containing identification and status data for each individual repair processed by the RMRS system. The record format consists of five fields, as follows:

FIELD#	FIELD NAME	FIELD DESC
1	REPID	RMRS system-assigned repair id number
2	FACNUM	Airbase facility number designation
3	PRIORITY	Airbase-assigned mission critical facility priority
4	SCHDPRTY	RMRS system scheduling priority (initially equal to (3) selectable)
5	STATUS	RMRS system repair status

The "repair id" (Field 1) is assigned sequentially by the RMRS system at repair load-time by the RMRS system (initial phase of PROCESS cycle). Although the "repair id" is mainly used internally by RMRS, it is provided to the user along with other repair information when appropriate. Most importantly, this "repair id" provides the required link between the repair and its associated resource requirements and allocations.

The "facility number designation" (Field 2) is populated at repair load-time. The facility number is extracted from the PDES output file name (e.g., B4058.EQP -> 4058).

The "mission critical facility priority" (Field 3) is the priority corresponding to the Airbase priority listing for mission critical facilities. This data is retrieved from the facility priority database, FACPRT.DBF.

The "scheduling priority" (Field 4) allows user control over repair ordering in the prioritized RMRS scheduler queue. By default, the "mission critical facility" and "scheduling priority" are the same, the scheduler *always* looks at the "scheduling priority" when ordering the queue. Therefore, the user can manipulate the scheduling order by using the SET PRIORITY menu function to change the value in Field 4, while maintaining the original priority in Field 3.

The "RMRS system repair status" (Field 5) is denoted by a single character designation that indicates the repair status as related to the RMRS processing cycle. The possibilities are as follows:

N - (N)ew repair
P - (P)ossible repair
Q - (Q)ueued repair
C - (C)ompleted repair
O - (O)verriden repair
S - (S)uspended repair
X - (X)canceled repair

5. EQPREQ.DBF

The EQPREQ.DBF database consists of records (one or more per repair) containing equipment requirement and time-slot allocation information associated with a repair. Consequently, the actual schedule information is embedded in this database. Each individual equipment requirement for a repair is assigned a record that is divided into six fields:

FIELD#	FIELD NAME	FIELD DESC
1	EQPID	Equipment id number
2	EQPDESC	Textual equipment description
3	START	Scheduled repair start time (in seconds referenced from Greenwich Mean Time 00:00:00 1/1/70)
4	DURATION	Estimated repair duration in seconds
5	REPID	Associated RMRS repair id number
6	STATUS	RMRS system allocation/repair status

The "equipment id" (Field 1) corresponds to the RMRS system-assigned number used in the Airbase equipment supply database, EQPSUP.DBF.

The "equipment description" (Field 2) is populated at repair load-time with the corresponding entry from the *.EQP PDES output facility file. The RMRS system currently uses this field as the search key when checking for equipment existence in the Airbase equipment supply database, EQPSUP.DBF.

The "scheduled repair start time" (Field 3) is set when a repair is scheduled. Its value is represented in seconds, referenced from Greenwich Mean Time (GMT) 00:00:00 1/1/70. Every equipment requirement record for a given scheduled repair will have the same start time value.

The "estimated repair duration" (Field 4) is populated at repair load-time with the corresponding entry from the *.EQP PDES output facility file. Its value is represented in seconds.

The "repair id" (Field 5) provides the link between the particular equipment requirement and its corresponding repair record in REPAIR.DBF.

The "RMRS system scheduling/repair status" (Field 6) is denoted by a single character designation that indicates the equipment requirement status as related to the RMRS equipment scheduling/repair processing scheme. The possibilities are as follows:

N - associated with (N)ew repair.
Q - (Q)ueued (associated with UNLOCKED (Q)ueued repair).
L - (L)ocked (associated with LOCKED (Q)ueued repair).
C - associated with (C)ompleted repair.
O - associated with (O)verriden repair.
S - associated with (S)uspended repair.
X - associated with (X)canceled repair.

6. MATREQ.DBF

MATREQ.DBF consists of records (zero or more per repair) containing material requirement and allocation status information associated with a repair. Each individual material requirement for a repair is assigned a record that is divided into six fields:

FIELD#	FIELD NAME	FIELD DESC
1	MATID	Material id number
2	MATDESC	Textual material description
3	QTY	Quantity required
4	UNIT	Textual "units" description
5	REPID	Associated RMRS repair id number
6	STATUS	RMRS system allocation/repair status

The "material id" (Field 1) corresponds to the RMRS system-assigned number used in the Airbase material supply database, EQPSUP.DBF.

The "material description" (Field 2) is populated at repair load-time with the corresponding entry from the *.MAT PDES output facility file. The RMRS system currently uses this field as the search key when checking for material existence in the Airbase equipment supply database, MATSUP.DBF.

The "quantity required value" (Field 3) is populated at repair load-time with the corresponding entry from the *.MAT PDES output facility file.

The "'units' description" (Field 4) is populated at repair load-time with the corresponding entry from the *.MAT PDES output facility file. This value is not currently used by the RMRS system except as information output.

The "repair id" (Field 5) provides the link between the particular material requirement and its corresponding repair record in REPAIR.DBF.

The "RMRS system allocation/repair status" (Field 6) is denoted by a single character designation that indicates the material requirement status as related to the RMRS material allocation/repair processing scheme. The possibilities are as follows:

A - (A)llocated (associated with (P)ossible or UNLOCKED (Q)ueued repair).
L - (L)ocked (associated with LOCKED (Q)ueued.
U - (U)sed (associated with (C)ompleted repair).
N - (N)ew (associated with (N)ew repair).
O - associated with (O)verriden repair.
S - associated with (S)uspended repair.
X - associated with (X)canceled repair.

7. REPINFO.DBF

REPINFO.DBF consists of records (one per repair) containing detailed information associated with a repair. The information is assembled from the data contained in the *.OUT PDES output file. Each individual repair is assigned a repair information record that consists of six fields:

FIELD#	FIELD NAME	FIELD DESC
1	REPID	Associated RMRS repair id number
2	FACFUN	Facility function
3	FACDESC	Facility description
4	ELENUM	Element number
5	ELEDESC	Element description
6	DAMMODE	Damage mode
7	DAMW	Damage width
8	DAML	Damage length
9	DAMH	Damage height
10	REPSTGY	Repair strategy
11	REMARKS	Remarks

The "repair id" (Field 1) provides the link between the particular repair information record and its corresponding repair record in REPAIR.DBF.

Fields #2-#11 are self-explanatory.

C. DATA FLOW

1. General

As a repair moves through the RMRS system it progresses from one stage, or status, to the next. Although conceptually a repair "moves" through the system, repair data is not actually copied from one place to another. Once repair data is initially recorded into the appropriate databases, the repairs status changes via the value of its status field as shown in Figure 2.

2. New Status

When the PDES output facility files (*.EQP, *.MAT, *.OUT) are loaded into the four repair databases, file format validation is performed and the new repair data records are

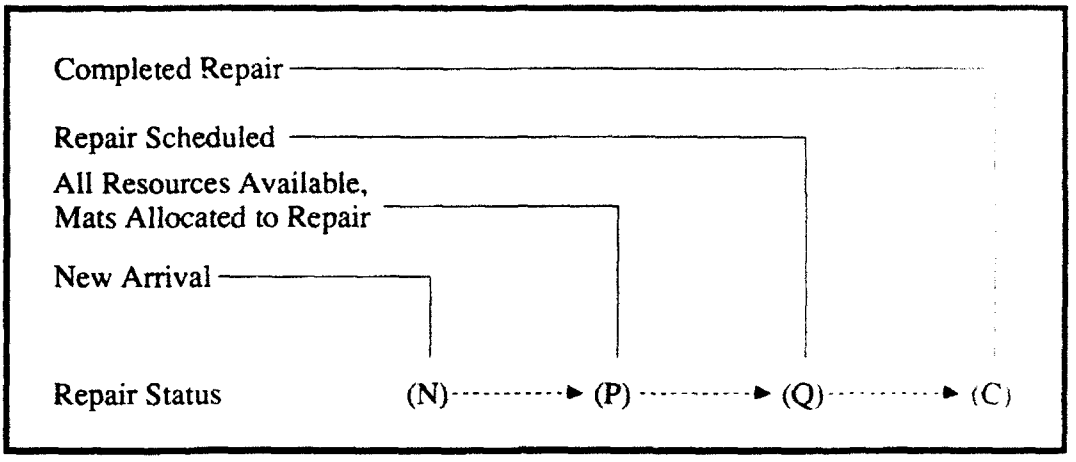


Figure 2. Repair Status Values.

marked with the (N)ew status. Typically the next status for a repair after (N)ew is (P)ossible, however there are other possibilities based on existing conditions and user decisions. Listed below are the status transitions that can occur from (N)ew:

(N)ew -> (X)canceled : NONEXISTENT EQUIPMENT. Repair required on a non-existent piece of equipment (for supply purposes, e.g., it was destroyed in the attack), the repair would be "automatically" (X)canceled by the RMRS system (RMRS assumes that only material conflicts can be resolved through substitution, equipment conflicts are not resolvable, however the user may perform manual resolutions).

(N)ew -> (S)uspended : ONLY MATERIAL CONFLICTS. Repair has one or more material requirement conflicts. A material conflict arises when a material proposed by the PDES module is either non-existent (no corresponding entry in Airbase material supply listing) or the Airbase material supply lacks sufficient quantities to satisfy the requirement. The RMRS system incorporates provisions to resolve material conflicts through substitution, according to the user's discretion. However, the user can defer this resolution opportunity to a later time by choosing not to attempt a conflict compromise. In this case, the RMRS system would (S)uspend the repair.

(N)ew -> (P)ossible : NO RESOURCE CONFLICTS. This status progression occurs when no resource conflicts arise.

3. Possible Status

A (P)ossible status indicates that the repair is ready for scheduling, the required equipment resources are available, and the required material resources have been allocated.

(P)ossible -> (Q)ueued : NORMAL SCHEDULING. This status progression occurs upon normal scheduling of the repair.

(P)ossible -> (S)uspended : MANUAL SUSPENSION. This status progression occurs when, before being scheduled, a repair with no resource conflicts is selected for suspension by the user.

(P)ossible -> (O)verriden : PRIORITY OVERRIDE RESOLUTION. This status progression occurs when, before this particular repair was scheduled, a different, higher-priority repair needs materials that are not available in the supply, but can be taken from this (P)ossible repair. In this case, the RMRS system asks the user for override permission which, if granted, will cause the required materials to be released to the high-priority repair, while the other materials from the overridden repair return to the material supply.

(P)ossible -> (X)cancel : MANUAL CANCELATION. This status progression occurs when, before being scheduled, a repair with no resource conflicts is cancelled by the user.

4. Queued Status

A queued status indicates that the repair has been scheduled and that the required equipment resources have been reserved for the appropriate time.

(Q)ueued -> (C)omplete : NORMAL COMPLETION. This status progression occurs upon normal completion of the repair.

(Q)ueued -> (S)uspended : MANUAL SUSPENSION. This status progression occurs when, before being started, a scheduled repair is selected for suspension by the user.

(Q)ueued -> (O)verriden : PRIORITY OVERRIDE RESOLUTION APPRVD. This status progression occurs when, before this particular repair begins, a different, higher-priority repair needs materials that are not available in the supply, but can be taken from this (Q)ueued repair. In this case, the RMRS system asks the user for override permission which, if

granted, will cause the required materials to be released to the high priority repair while the other materials from the overridden repair return to the material supply.

(Q)ueued -> (X)cancel : MANUAL CANCELATION. This status progression occurs when, before being started, a scheduled repair is cancelled by the user.

5. Complete Status

Once a repair has acquired the (C)omplete status it remains in the database files until explicitly deleted or the database files are reset.

D. RESOURCE ALLOCATION

Resource allocation by the RMRS system is accomplished automatically through software manipulation of the Airbase supply databases and the repair resource requirement databases. The two resource types, equipment and material, are allocated in separate phases of the RMRS processing cycle due to the the different concerns for each type of resource. For material resources we are mainly concerned with quantity available; while for equipment resources we are concerned with the equipment allocation over time. The Process menu commands accomplish the material resource allocation, while the Schedule menu commands handle the allocation of equipment resources.

Material resource allocation is automatic through the normal execution of the RMRS Process commands: (1) Auto mode; (2) Single facility; and (3) Compromise retry.

The *Process / Auto Mode* and *Process / Single Facility* commands involve the processing of newly arrived PDES facility repair file sets. This processing includes database population, resource availability analysis, and resource requirement compromise, if needed. The *Process Compromise Retry* command involves availability analysis and compromise. Regardless of the command used, the overall objective here is allocate material resources to a repair if no conflicts occur or if those occurring can be resolved.

The Process menu commands work on an "all or nothing" resource availability analysis scheme. This scheme verifies that all the resource requirements can be satisfied before resource allocation actually occurs since, for most repair strategies, it is unlikely that the repair could be satisfactorily completed with only a fraction of the required resources. Thus, the Process menu commands verify that the required equipment exists and then that the appropriate material quantities

are available. Then, upon no conflicts or upon resolution thereof, the materials are allocated to that repair. The equipment is allocated at schedule time.

The RMRS system requirement of prioritized resource allocation is provided for in two ways. First, all newly arrived PDES facility repair file sets are processed through resource availability analysis in priority order. If all resources are readily available from the Airbase supply databases, then the material resources are allocated at that time. Second, when a material conflict occurs because of inadequate supply, the RMRS system turns to the material requirements database (MATREQ.DBF) to attempt to locate a priority-override resolution. In other words, the search tries to locate a material requirement that has sufficient quantity and is associated with a repair that has a lower priority than that repair looking for resources. This search only includes the matching material requirements for repairs that have not been started yet. Specifically, these qualifying material requirements will have a STATUS designation of (A)llocated. In contrast, unallocated material requirements will be marked (S)uspended or (X)canceled, and those associated with a completed repair are marked (U)sed. When used in conjunction, these two methods assure that the material resources are efficiently distributed in a prioritized fashion, as required.

On the data manipulation level, material allocation involves both the Airbase material supply database (MATSUP.DBF) and the material requirements database (MATREQ.DBF). In a no-conflict scenario, the material quantities specified in the repair's material requirement records are simply deducted from the QTY field in the AB material supply database, and the corresponding material requirement records are given a STATUS of (A)llocated. Of course, a side effect of this transaction is that the associated repair identification record (in REPAIR.DBF) has its STATUS field changed to (P)ossible. Whereas, with the override resolution scenario, the overridden repair is dropped from its scheduled status of (Q)ueued. This must be the case, because once the required override materials have been taken from the repair, it no longer has access to all the material resources it needs. Upon demotion of the subordinate repair, all of its allocated material resources are first returned to the AB material supply by adding the requirement quantities to the QTY fields in the corresponding supply entries. The overridden repair and its associated resource requirement records are given a STATUS of (O)verridden. Then the materials are allocated to the high-priority repair in the normal way.

E. REPAIR SCHEDULING

Once resource allocation has been successfully completed through the use of one of the Process menu commands, the REPAIR.DBF database file should contain one or more records with the status field set to (P)ossible. A (P)ossible status indicates that the repair has been allocated

material resources and is ready to be scheduled. Remember that the resource allocation cycle verified that the equipment required for each repair existed, therefore the only concern at this point is to find the earliest common time that all pieces of equipment required for the repair will be available. When the time slot is found, the status field in REPAIR.DBF is set to "Q" for "queued," which means that the repair has been scheduled and equipment allocated. Also the status, start, and duration fields in the EQPREQ.DBF file are set to the appropriate values to indicate that the time slot for that piece of equipment has been allocated.

As an example of the way RMRS schedules repairs, consider the case where we schedule one repair with a duration of one hour that requires the following pieces of equipment: one Shotcrete Machine, two Ramsets, and one Repair Team. Assume that other repairs have already been scheduled and that the current time is 1200 hours today. To begin the scheduling process RMRS will first determine the equipment requirements for the repair (which we already know from the problem statement) from the EQPREQ.DBF file. Next, the earliest start time for the repair is determined by adding an appropriate lag value to the current time, for example thirty minutes; therefore, we will not attempt to schedule this repair before 1230 hours today. Now we choose the first of the equipment pieces (the selection order is not important) and try to locate a one-hour time slot (one-hour duration repair) when this piece of equipment is available not before 1230 hours today (earliest start time). The beginning of the time slot we find is called the "equipment start time." We repeat the process for each piece of equipment. Depending on the current availability for each piece of equipment, the equipment start times may or may not be the same. If the start times are the same we have scheduled the repair, however if they are different we must try again. For the retry we set the earliest start time for the repair to the latest of all the equipment start times, and repeat the process. Eventually we will find a common time during which all of the equipment pieces required for the repair will be available. The above approach ensures that the common time is the earliest possible.

SECTION III

COMMAND REFERENCE

The Resource Manager/Repair Scheduler (RMRS) operates in a pull-down menu environment. All menu choices can be selected by mouse or via the keyboard. The system menu bar consists of the following six pull-down menus:

(1) SYSTEM
(2) INIT/SETUP
(3) PROCESS
(4) SCHEDULE
(5) CHANGE STATUS
(6) GEN REPORT

The following is a brief description of each menu bar selection:

A. SYSTEM MENU (ALT-Y) [2]

This menu allows the user to access information about the software release and view/modify the raw data contained in the RMRS system databases, along with the conventional operating system access functions that allow DOS shell operations and program termination.

About. Choosing this command will cause the appearance of a dialog box that shows version information for the RMRS system software. Press any key to close the box.

Edit Data (Alt-E). The *System / Edit Data* command lets you view and/or modify the raw data contained in the seven RMRS databases. No special administrator privileges are needed to modify the databases at this time. However, since the integrity of the RMRS system output depends entirely on that of the data contained within these databases, future system enhancements should include some security mechanism that regulates the data modification privilege (e.g., a password). For a description of the features available in the datafile edit module consult the online help.

DOS shell (Alt-D). The *System / DOS Shell* command lets you temporarily exit the RMRS system to enter a DOS command or program. To return to the RMRS system, type 'EXIT' and press <ENTER>.

[2]Some menus and menu choices have short cut keys. These keys are indicated in parenthesis.

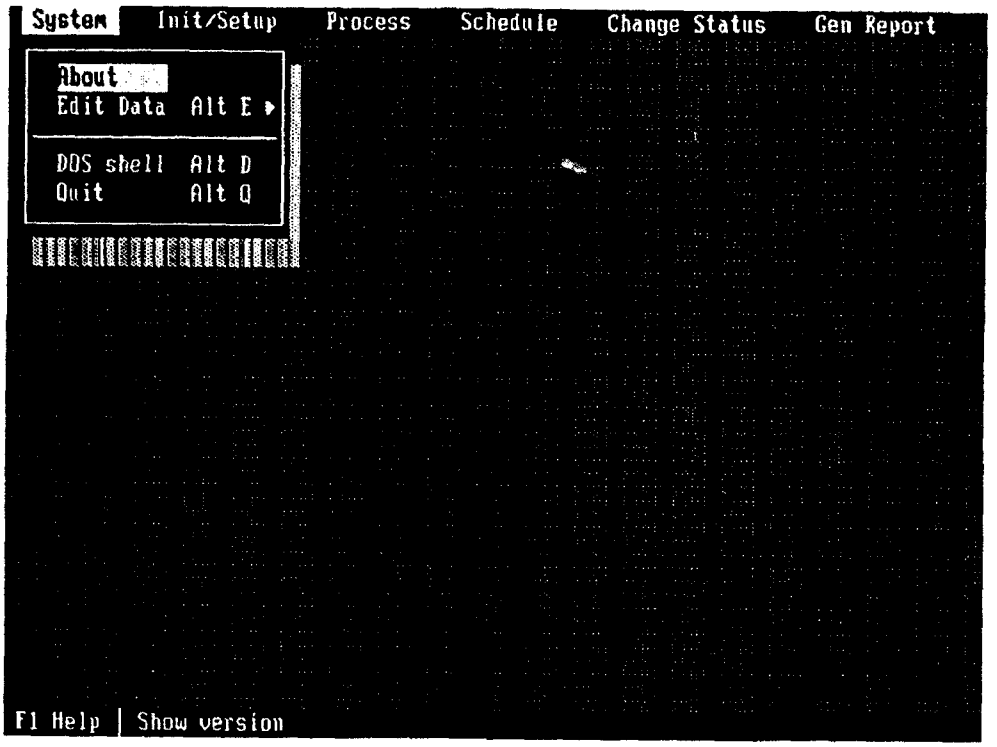


Figure 3. System Menu.

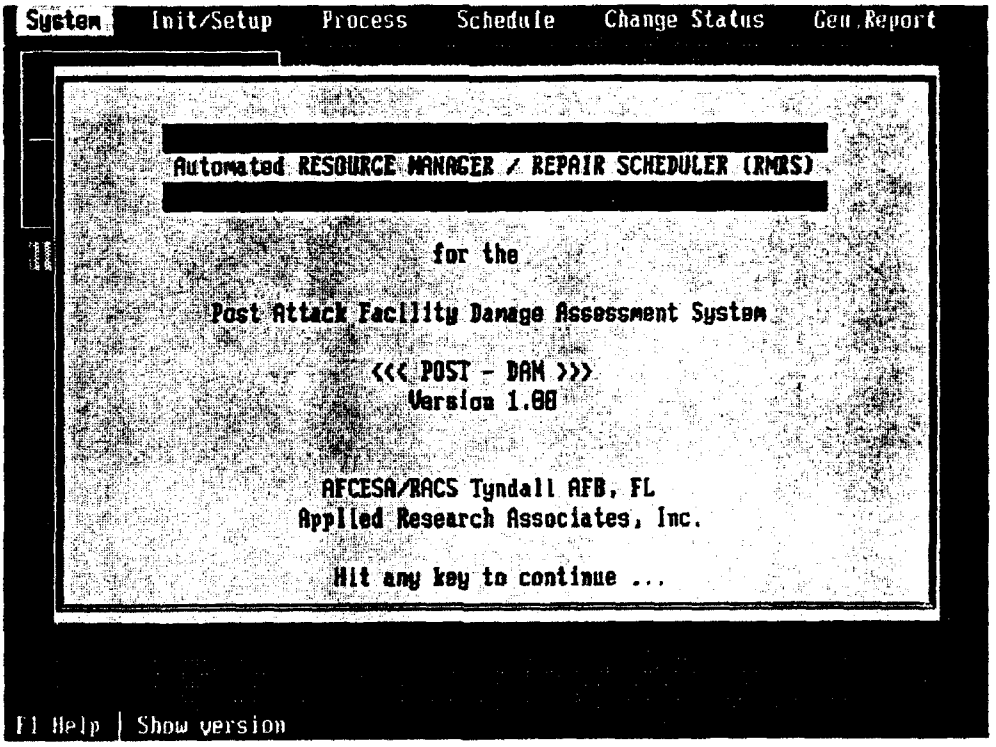


Figure 4. About Screen.

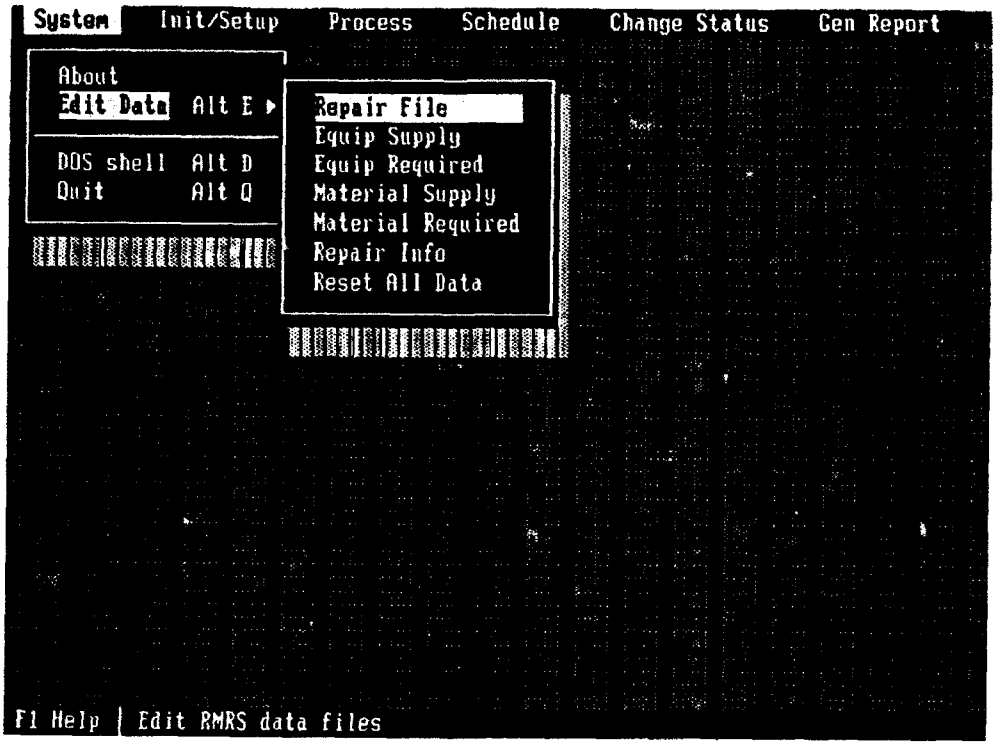


Figure 5. Edit Data Menu.

Quit (Alt-Q). The *System / Quit* command, upon verification, exits the RMRS system program, removes it from memory, and returns you to the DOS command line.

B. INIT/SETUP MENU (ALT-I)

This menu allows the user to load new supply files after program start-up.

Load new AB supply file. The *Init/Setup / Load New Supply File* command lets you convert a raw Airbase supply file to its corresponding RMRS system database. This command selects a secondary pulldown menu presenting two choices: AB materials supply file, and AB equipment supply file. Both commands open a dialog box that will accept the complete path/filename specification for the supply file to be loaded.

C. PROCESS MENU (ALT-P)

This menu allows the user to load newly arrived PDES output repair files and, subsequently, allocate material resources to a repair, barring any unresolved resource conflicts, thereby generating possible repairs to be passed to the RMRS scheduler.

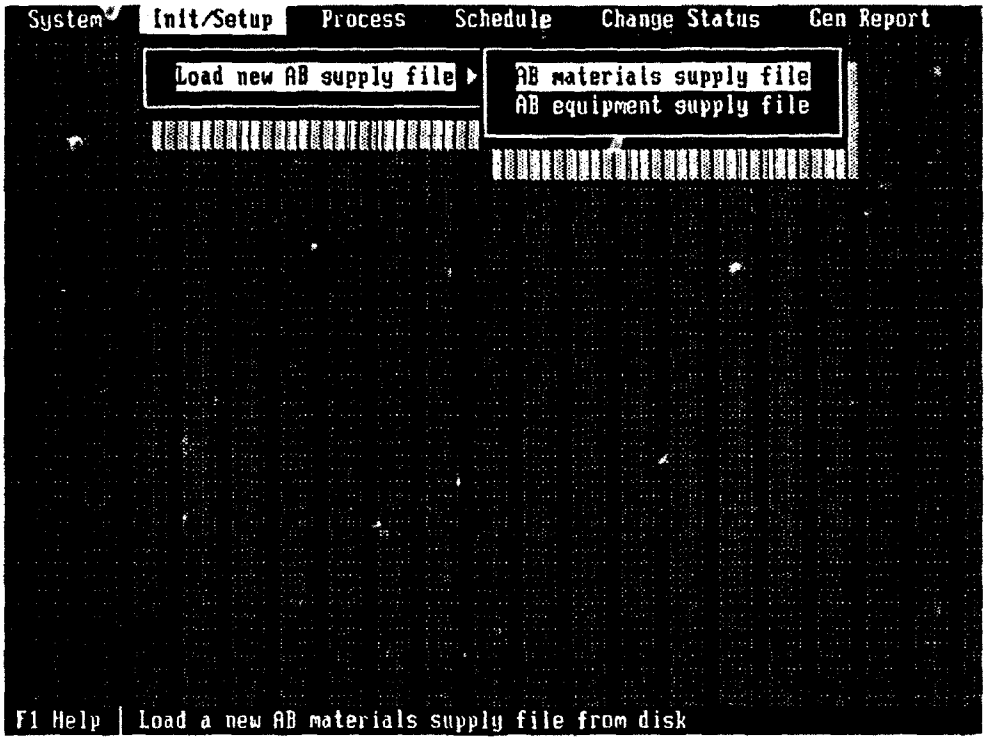


Figure 6. Init/Setup Menu.

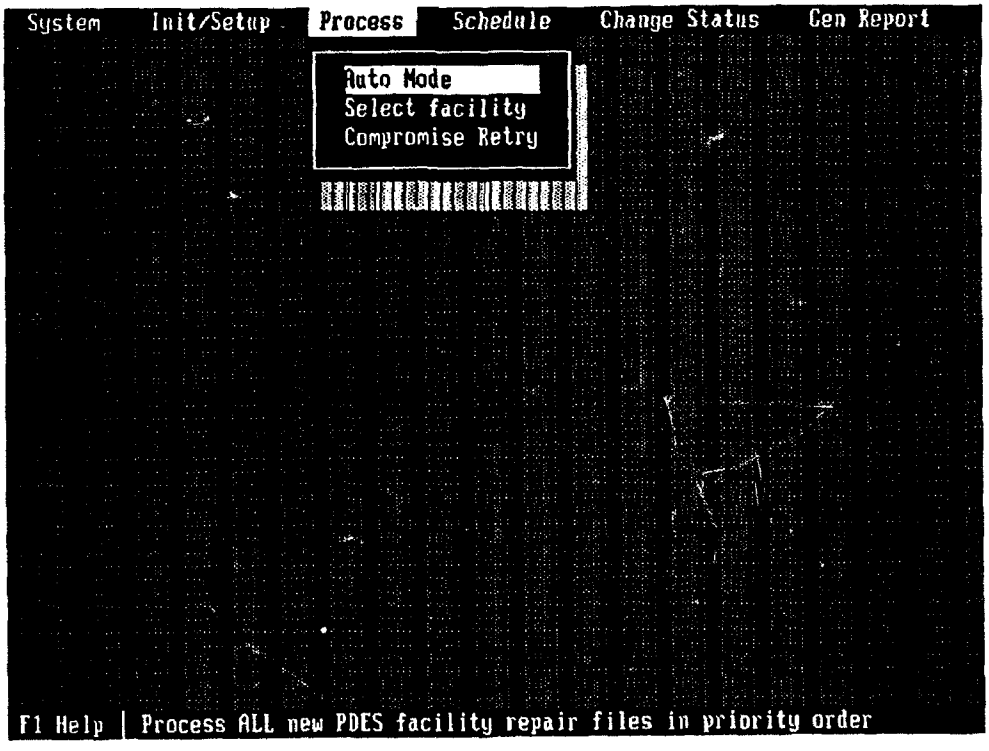


Figure 7. Process Menu.

Auto mode. The *Process / Auto Mode* command lets you (1) "process" *all* of the newly arrived PDES facility repair files; (2) "process" load repair data into RMRS databases from PDES .EQP, .MAT, and .OUT files; and (3) allocate material resources to those repairs if either no resource conflicts occurred or all conflicts were resolved.

"Newly arrived" PDES files are those found in the PDES Input Directory. After "processing," the PDES files (.EQP, .MAT, and .OUT files) are placed in a PDES Archive Directory. These archived copies are kept as a safety precaution because the corresponding copies in the PDES Input Directory are removed after the RMRS system 'Process' phase.

Select. The *Process / Select Facility* command lets you "process" only the facilities that you select from the newly arrived PDES facility repair files. It displays a file-selection dialog box for you to select the facility repair files that are to be "processed."

The dialog box contains a file list of .EQP files corresponding to each available PDES facility repair file set. Press the UP/DN arrows to peruse the list and press <ENTER> to mark a facility file for selection. When finished simply type Ctrl-<ENTER> to accept or <ESC> to cancel.

Compromise. The *Process / Compromise Retry* command lets you attempt to retry the resolution of the material resource conflicts associated with a (S)uspended or (O)verridden repair. It is called "retry" because when a resolvable resource conflict arises during the "process" stage, the user is alerted and given the opportunity to initiate a "Compromise" session. It first displays a dialog box with a complete list of all the repairs with (S)uspended or (O)verridden status. It then allows the user to select the repair with which to work.

Upon selection, a "Compromise" session is initiated that presents the needed information to make material resource substitution decisions. The session window contains a repair data box, a conflict count status box, a scrollable material conflict list, a scrollable material supply listing, a step-by-step instruction box, and several selection display boxes.

D. SCHEDULE MENU (ALT-S)

This menu allows the user to manipulate scheduling criteria, if desired, and to schedule the repairs that are possible.

Full. The *Schedule / Full Schedule* command lets you generate a new schedule with regard only to repair priority. Repairs that may have been previously scheduled and have not begun are rescheduled along with newly arrived repairs.

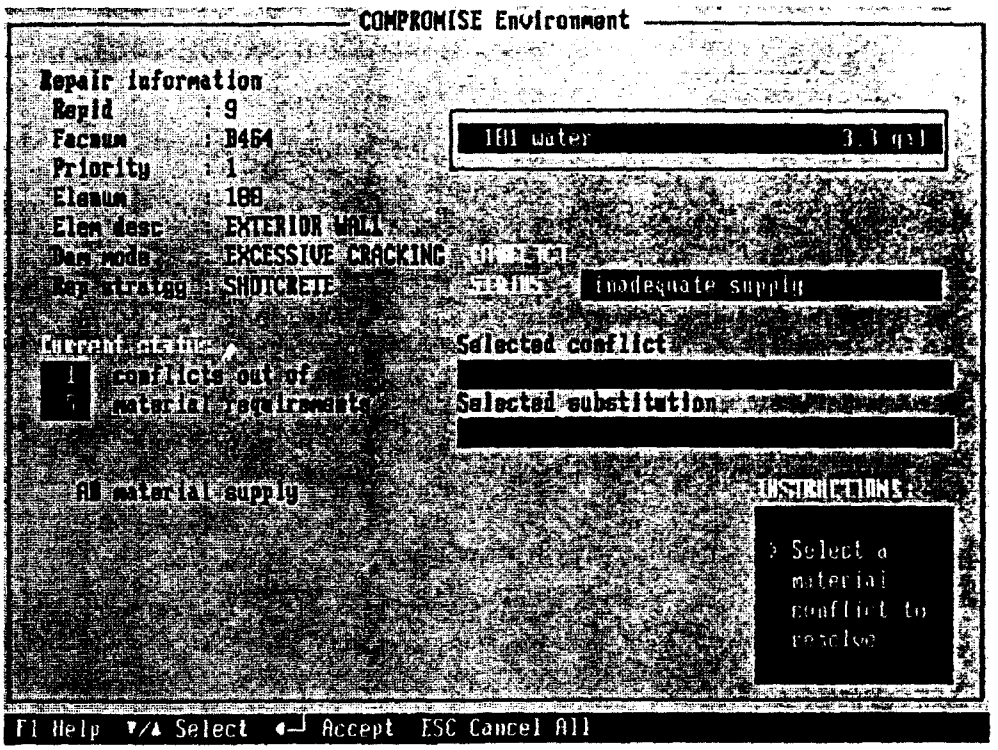


Figure 8. Compromise Dialog Box.

Figure 9. Schedule Menu.

Update. The *Schedule / Update Schedule* command lets you generate an amended schedule in which previously scheduled repairs retain their time slot and new repairs are scheduled on a priority order basis.

Set. The *Schedule / Set Priority* command lets you change the scheduling of a repair. This allows manipulation of the prioritized scheduling queue order. This command displays an input screen that allows the user to select the repair of interest and specify the new priority.

E. CHANGE STATUS MENU (ALT-C)

This menu allows the user to designate a job in the repair possible list as completed or canceled.

Completed. The *Change Status / Completed* command lets you change the repair status of a (Q)ueued repair to (C)ompleted. This should be used when notification has been received that a repair has been completed.

Canceled. The *Change Status / Canceled* command lets you change the repair status of a (Q)ueued or (S)uspended repair to (X)canceled. This should be used when notification has been received that a repair has been canceled.

NOTE: The associated material resource requirements (if any) that were previously allocated, will be returned to the Airbase material supply database. Consequently, any reactivation of that repair would require that it again be "processed" and scheduled as before.

F. GEN REPORT MENU (ALT-R)

This menu allows the user to generate repair schedules and reports.

Summary. The *Gen Report / Summary Schedule* command generates an abbreviated format repair schedule which contains the "repair id" and description, repair start and completion times and overall duration for all repairs scheduled. This information is presented to the user on the screen and also output to a disk file named SUMMARY.REP.

Detailed. The *Gen Report / Detailed Schedule* command generates a schedule with same information as the summary schedule and also includes more detailed information about the repair such as equipment and material requirements, facility priority, and overall repair statistics such as number of repairs scheduled, number of repairs completed, etc. This information is presented to the user on the screen and also output to a disk file named DETAILED.REP.

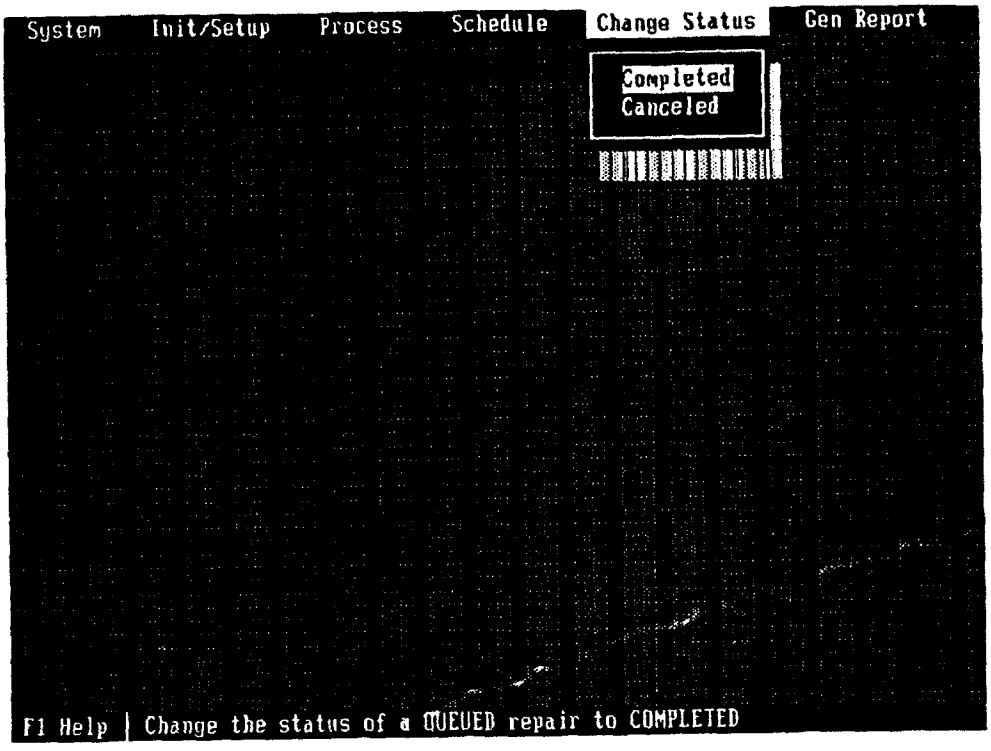


Figure 10. Change Status Menu.

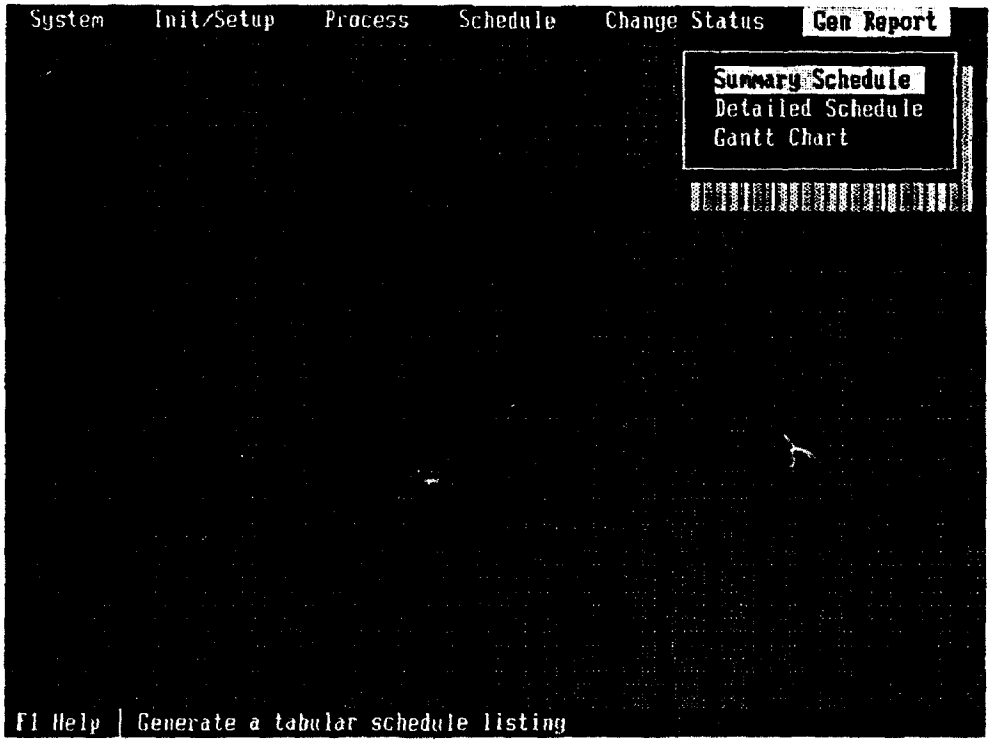


Figure 11. Gen Report Menu.

Gantt. The *Gen Report / Gantt Chart* command generates a Gantt chart that allows the user to quickly visualize the temporal distribution of the repair schedule. This information is presented to the user on the screen and also output to a disk file named GANTT.REP.

SECTION IV

EXAMPLE SESSION

This section presents an example case and its corresponding RMRS operations in order to acquaint the user with the operation of the RMRS system.

Our example case involves two PDES Input facility repair file sets -- one for facility B138 and one for facility B4058 and assumes the RMRS system repair-specific databases are not empty (i.e., some repairs have been processed by the RMRS system). These file sets have just arrived from the POST-DAM Expert System (PDES). At this point, the PDES has evaluated the on-site damage reports from the Damage Assessment Teams (DATs). Its chosen repair strategy is outlined in the standard system facility repair file set, consisting of a required equipment listing (.EQP), a required material listing (.MAT), and a repair information report (.OUT). Figure 12 shows the RMRS startup as displayed when the system is started.

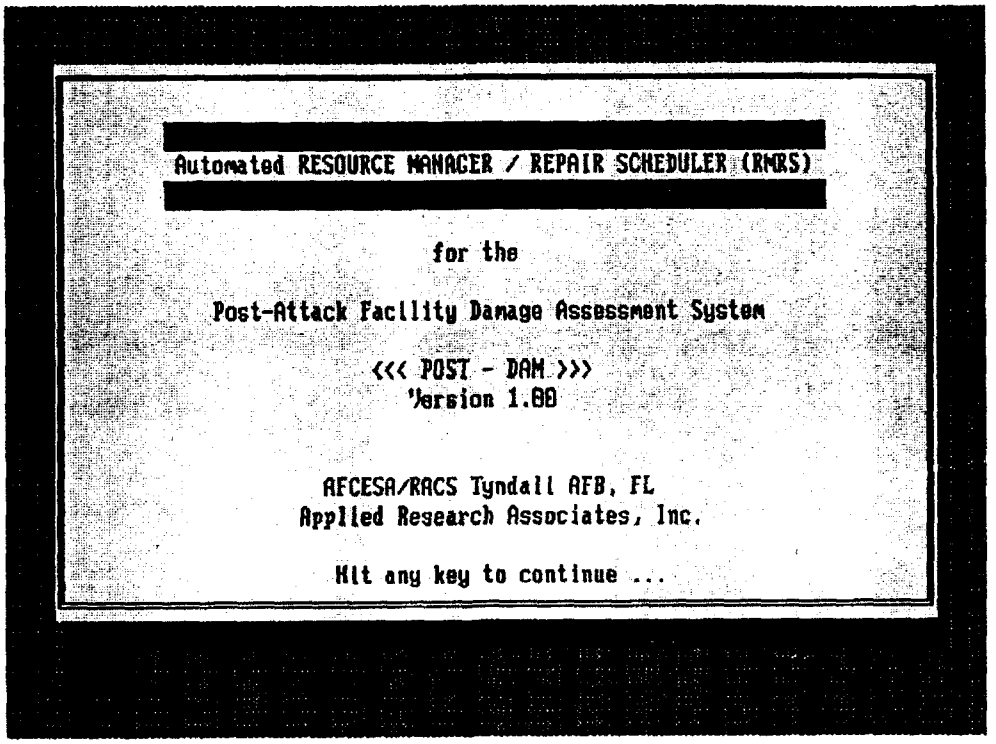


Figure 12. Startup Screen.

A. REPAIR LOADING | PROCESSING

To initiate the RMRS processing of these two facility repair file sets, select either the *Process / Auto Mode* or the *Process / Select Facility* command (i.e. the *AutoMode* or *Select Facility*

choice from the *Process* menu). In this case, we would like to process both facility file sets in sequence, so we select *Process / Auto Mode* as shown in Figure 13. On the other hand, to process only one facility, the *Process / Select Facility* command should be selected. Both of the *Process* menu commands perform three tasks:

(1) Load new repair data
(2) Resolve resource conflicts if possible
(2) Allocate the appropriate material resources

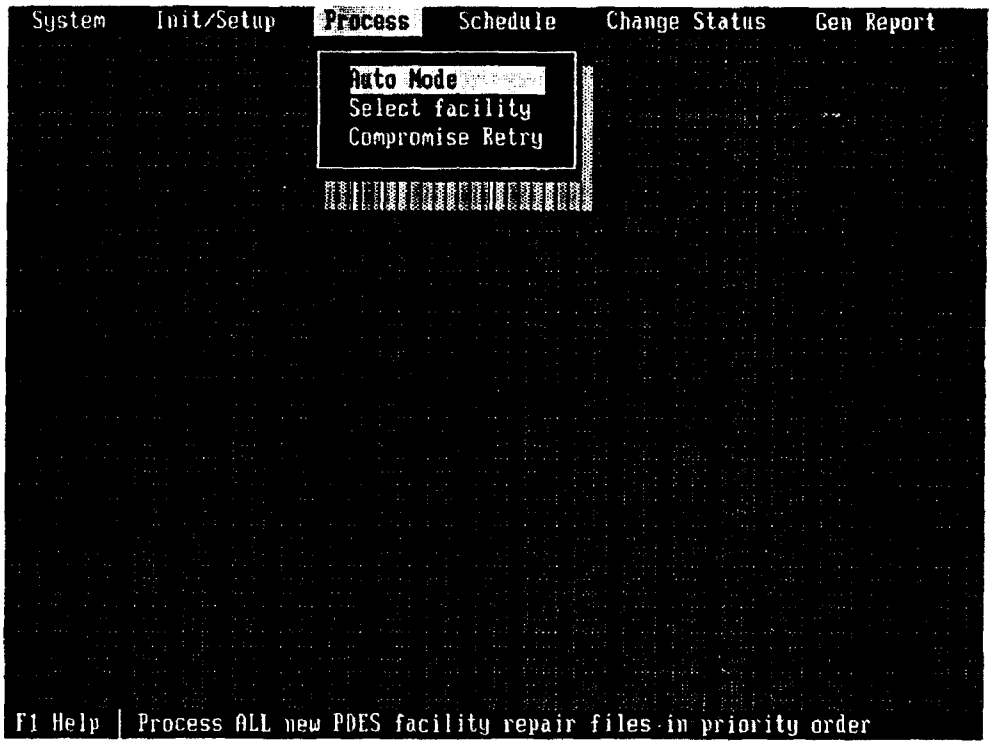


Figure 13. *Process / Auto Mode* Command.

After loading all new repairs, the *Process / Auto Mode* command sets up a prioritized repair queue from which repairs are individually processed to determine resource availability and are then allocated materials when no resource conflicts exists. In our case, facility 4058 has the higher priority and, consequently, its repairs are processed first.

The first repair for facility 4058 has been assigned a REPID of 105 by RMRS, and is evaluated for existence of resource conflicts as shown below.

The remaining repairs for facility 4058 are processed likewise. The equipment and material description strings serve as the search keys into the respective supply databases. If located, the

TABLE 1. RESOURCE AVAILABILITY ANALYSIS: REPID #105.

EQP requirements	Found ?	(✔-yes/X-no)
sc machine		✔
ramset		✔
ramset		✔
repair team		✔
MAT requirements	**Found ?**	**Enough avail ? (✔-yes/X-no)**
2x4 16ft (2 ea)	✔	✔
plywood 4x8 .5in (3 ea)	✔	✔
wire mesh (40.0 sqf)	✔	✔
shotcrete (5.9 cy)	✔	✔

"supply id" number associated with the material resource is added to the corresponding material requirement record to allow a more efficient numerical key search to be used in future searches. In the equipment requirement database, the "equipment id" number assignment is postponed until scheduling in order to address the situation of efficient time allocation for multiple supply listings of identical equipment pieces. When a resource match is not found in the appropriate supply file, that resource requirement constitutes one instance of a resource availability conflict (conflict, for short). All equipment conflicts are of this type. On the other hand, a material conflict can also arise from the situation where the material is listed in the supply, but the quantity available does not satisfy the requirement as specified.

A repair that has resource conflicts is handled in one of two ways, depending upon the types of conflicts involved. If the repair has one or more equipment conflicts, that repair is automatically (X)canceled. However, the user is free to make any manual resolution he desires. On the other hand, if the resource conflicts are limited to just materials, the user is alerted to this repair status and given the opportunity to enter the RMRS Compromise mode to make appropriate material substitutions towards a resolution. In addition, a priority override solution is also available for resolution of a material conflict, provided that the needed materials were previously allocated to a lower priority repair (that has not been started and is not (L)ocked in schedule) from which they can be stolen, if desired by the user.

For this example, no resource conflicts occurred for any of the repairs processed. The required materials for each repair were deducted from the available amounts in the material supply, the material requirement records were marked as (A)llocated, and the repair id records were marked as (P)ossible. Also, the equipment requirement records associated with these are left marked as (N)ew.

B. SCHEDULE REPAIRS

Having been Processed by the RMRS system, all (P)ossible repairs are now ready to be included in the schedule. At this point there are two scheduling options, *Full Schedule* and *Update Schedule*. The *Full Schedule* command generates a new schedule in which all previously scheduled repairs are rescheduled along with the newly arrived repairs. The *Update Schedule* command retains the current schedule and amends the newly arrived repairs. This example uses select the *Schedule / Full Schedule* command as shown in Figure 14.

Figure 14. *Schedule / Full Schedule* Command.

The actual database manipulation involved here occurs primarily within the equipment requirements database, EQPREQ.DBF. Each repair is taken in facility priority order and scheduled at the earliest possible time based on equipment availability. Upon completion, all of the repairs will have corresponding equipment requirement records possessing a valid "equipment id" number, a START time, and a (Q)ueued status. This information constitutes the repair schedule.

C. REPORT GENERATION

The RMRS system allows for the generation of several output reports including the Summary, the Detail, and the Gantt chart formats. For this case we will choose the *Gen Report Gantt Chart* menu choice as shown in Figure 15. Once this menu choice is selected a scrollable

report is displayed on the screen similar to the one shown in Figure 16. This report contains information on the repair scheduling and equipment allocation. This report is useful for viewing a schedule and then iterating on the results. A copy of the report also is also written to a file, named GANTT.REP, which can be output on a hard copy device.

Figure 15. *Gen Report / Gantt Chart* Menu Choice.

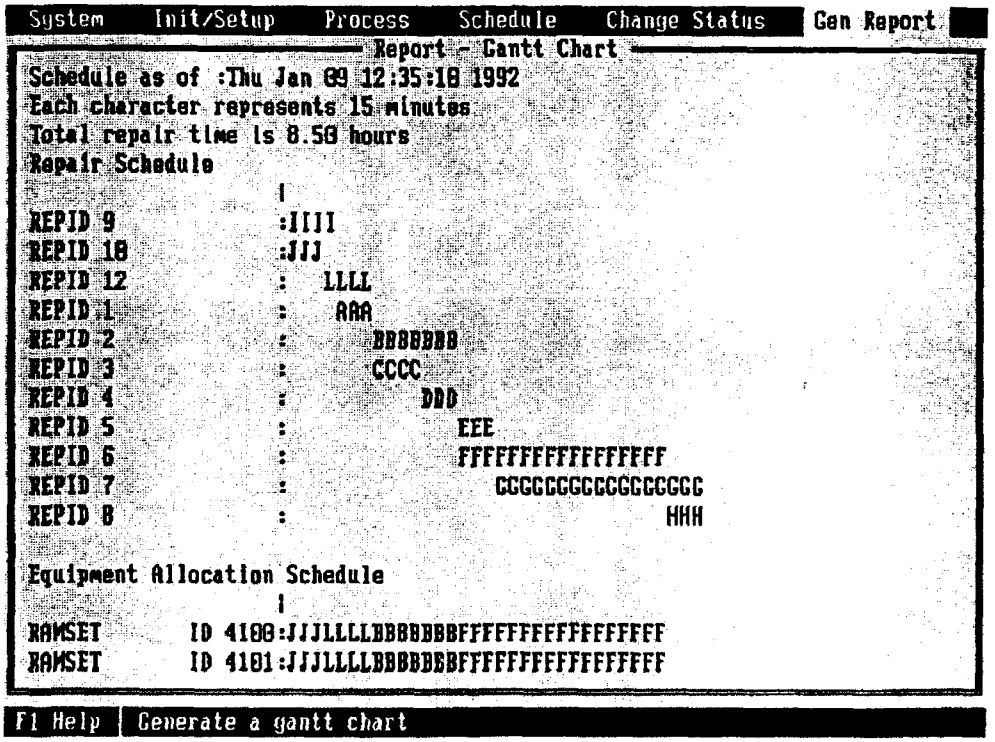


Figure 16. Gantt Chart Report.

SECTION V

CONCLUSIONS AND RECOMMENDATIONS

The RMRS prototype software clearly demonstrated an architecture that was able to quickly and efficiently manage Airbase resources and schedule expedient repairs for mission critical facilities. For repairs with no resource conflicts, the RMRS system could operate with minimal user intervention When conflicts were present, resolution was quick and simple in the compromise environment. By allowing the user unlimited access to all of the RMRS data files, complete control of the resource allocation and scheduling processes was possible.

For full-scale development of the RMRS software, the following issues should be addressed:

1. Include a communications module to allow RMRS to receive PDES data files in the background while continuing to work uninterrupted.

2. Incorporate Expert System capabilities to provide RMRS the capability to suggest resource conflict solutions and changes in repair strategies based on current resource supplied and equipment usage.

3. Develop an optimized scheduler to generate schedules based on not only facility priority but also time and resource optimization.

REFERENCES

1. *Postattack Damage Assessment of Facilities*, Subtask 2.02, Air Force Engineering and Services Center, SETA Contract F08635-88-C-0067, December 1987.

2. *Postattack Damage Assessment of Facilities*, Subtask 2.02.1, Air Force Engineering and Services Center, SETA Contract F08635-88-C-0067, October 1988.

3. *Postattack Damage Assessment of Facilities*, Subtask 2.02.2, Air Force Engineering and Services Center, SETA Contract F08635-88-C-0067, February 1989.

4. *The POST-DAM System, Volume 1, Introduction to the POST-DAM System*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, March 1991.

5. *The POST-DAM System, Volume 2, Software User's Manual for the Expert System*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, February 1991.

6. *The POST-DAM System, Volume 3, Software User's Manual for the DESQ view 386*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.

7. *The POST-DAM System, Volume 4, Software User's Manual for the Relational Data Base Management System*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.

8. *The POST-DAM System, Volume 5, Software User's Manual for the Harvard Project Manager*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.

9. *The POST-DAM System, Volume 6, Software User's Manual for Crosstalk Mk.4 on the Host Computer*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.

10. *The POST-DAM System, Volume 7, Software User's Manual for the TED 1.1 Editor*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, December 1990.

11. *The POST-DAM System, Volume 8, Software User's Manual for Crosstalk Mk.4 on the Remote Computer*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, March 1990.

12. *The POST-DAM System, Volume 9, Fixed Manual of Mission-Critical Facilities for Use with the Prototype POST-DAM System*, Applied Research Associates, Inc., ESL-TR-91-22, AFESC/RDCS, March 1991.

13. *Code Base 4.2*, Sequiter Software, Inc., 1988-1990.

14. *Greenleaf Data Windows, Volume I, Version 2.20*, Greenleaf Software, Inc., 1983-1990.

15. *Greenleaf Data Windows, Volume II, Version 2.20*, Greenleaf Software, Inc., 1983-1990.

APPENDIX A

INSTALLATION PROCEDURE FOR RESOURCE MANAGER/REPAIR SCHEDULER (RMRS)

I. INSTALLATION

A. System Requirements

The Resource Manager/Repair Scheduler (RMRS) is designed to run on the IBM PC family of computers, including the XT and AT, along with all true IBM compatibles. The following "minimum" hardware configuration has been established:

RAM storage: 640k

Hard Disk storage: 2 mb free (disk space limits number of repairs)

Peripherals:

(1) color or monochrome text display adaptor and monitor
(1) 5 1/4 or 3 1/2 floppy disk drive
(1) line printer

B. Installation Procedure

The Resource Manager/Repair Scheduler (RMRS) comes with an automatic installation program called INSTALL. INSTALL automatically copies the RMRS files to the appropriate directories on your hard disk drive. For reference, the README file on the installation disk includes a list of the distribution files along with recent information about RMRS.

To install RMRS:

(1) Insert the installation disk into drive A

(2) Type A:INSTALL <return>

(3) Follow the prompts

Note: If your installation drive is drive B, substitute B for A in steps 1 and 2.

APPENDIX B

RMRS SOURCE CODE

I. RMRS SOURCE CODE

RMRS was developed using the C Language and two subroutine libraries. Borland C++ 2.0 was chosen as the development environment because of availability and favorable past experience. Although this compiler supports a full implementation of AT&T's C++ version 2.0, as well as ANSI C, only ANSI C was used to develop RMRS. Greenleaf DataWindows© version 2.12 was used to develop the user interface, and Code Base 4.2© was used for the data file management. These libraries were chosen for their maturity and because both libraries have UNIX implementations. Both the DataWindows and Code Base license agreement allow for royalty-free distribution of applications, such as RMRS, that contain object code from their libraries. However, distribution of the libraries is prohibited. Thus, any person wishing to make modifications to the RMRS program will be required to obtain both DataWindows 2.12 and Code Base 4.2. Both of these packages are readily available at minimal cost.

```
 1 /*
 2  *
 3  *     comprom.c
 4  *
 5  *        module to compromise repair material conflicts
 6  *
 7  *        DoCompromise()
 8  *
 9  *        LAST REV: 12/30/91
10  *
11  *
12  */
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <string.h>
17
18 /* toolkit header files */
19 #include "dw.h"
20 #include "pwrbase.h"
21 #include "color.h"
22 #include "lkumod.h"
23 #include "d4base.h"
24 #include "u4error.h"
25
26 /* custom header files */
27 #include "rmrsutil.h"
28 #include "sysdefs.h"
29 #include "procutil.h"
30 #include "miscutil.h"
31 #include "dbutil.h"
32
33 /* items that can be highlighted */
34 #define CONFSTAT 1
35 #define SELTCONF 2
36 #define SELTSUB  3
37 #define CURSTAT  4
38 #define INSTRCT  5
39
40 /* ctrl items that can be selected */
41 #define CONFSEL 1
42 #define SUPPSEL 2
43 #define SELTQTY 3
44 #define NOSELCT 4
45
46 /* key defs for lku return value */
47 #define CANCEL_SEL   -1
48 #define ABORTSUB_SEL -2
49
50 /* DoCompromise() support fns ***********************************************/
51
52 HWND initcompwin(void);
53 HWND initkeymapbar(void);
54 void delcompwin(void);
55 void delkeymapbar(void);
56 void setupwinformat(void);
57 void dsplyrepinfo(REPDATANODE *);
58 void dsplycompstatus(int);
59 void dsplyconfstat(LBOXSTA *);
60 void dsplycompsuccessbox(void);
61 void updtcompstatus(LISTITEM *);
62 char * getnextmatconf(REQLSTNODE **);
63 char * getfirstmatconf(REQLSTNODE *, REQLSTNODE **);
64 char * getsupprecstr(long);
65 int isoverrideposs(LBOXSTA *);
66 int asktoacceptoverride(void);
67 void hilite(int);
68 void unhilite(int);
69 void ctrlpass(int, int);
70 float getqty(void);
71 int verifyabort(void);
72 int chkqtyvssupp(float, long);
73 int askifcorrectqty(void);
74 void alerttoqtylimit(long);
```

```
75 void dsplysubaccepted(void);
76 void recordsub(LBOXSTA *, float, long);
77 void setovrdflag(LBOXSTA *);
78
79 /*-----------------------------------------------------*/
80
81 static HWND KeymapBar;
82 static HWND CompWin;
83
84
85
```

```
int DoCompromise(long repid, REQLSTNODE * reqlstptr)
   85 (
   86 /*
   87  * ARGUMENT
   88  *  (long)         repid     - repair id
   89  *  (REQLSTNODE *) reqlstptr - pointer to resource availability analysis list
   90  *
   91  * DESCRIPTION
   92  *  Allows the user to interactively attempt a conflict resolution strategy
   93  *  through material substitutions and priority overrides on a repair
   94  *
   95  * RETURNS
   96  *  (int) SUCCESS or FAILURE
   97  *
   98  * DATABASES AFFECTED (whether direct or indirect)
   99  *    matsup_dbf (matsup_ndx2 - MATID index       ) - (CS$) to be added
  100  *
  101  */
  102
  103     LISTITEM * conflboxlst;
  104     LBOXSTA * conflbox;
  105     REQLSTNODE * currptr;
  106     char * strptr, * choice;
  107     REPDATANODE * repinfoblk;
  108     long selrecno;
  109     char * supprecstr;
  110     bool success = FALSE, acceptovrd = FALSE, validqty, acceptqty;
  111     /* these will be running totals */
  112     int numofmatreqs = 5;
  113     unsigned int key;
  114     float subqty;
  115
  116
  117     if (iseqpconf(reqlstptr))
  117     {
  118         u4error(INVCOMPCALL, "Invalid call to COMPROMISE", "function: ",
  119             "NO CONFLICTS DETECTED !", (char *) 0);
  120         return (-1);
  121     }
  122     KeymapBar = initkeymapbar();
  123     CompWin = initcompwin();
  124     repinfoblk = GetRepData(repid);
  125     dsplyrepinfo(repinfoblk);
  126     dsplycompstatus(numofmatreqs);
  127     setupwinformat();
  128
  129
  130
  131     /* load list box linked list with conflicting mat record strings */
  132     conflboxlst = initlist();
  133     strptr = getfirstmatconf(reqlstptr, & currptr);
  134     while (strptr != NULL)
  134     {
  135         adtolist(conflboxlst, strptr);
  136         strptr = getnextmatconf(& currptr);
  137     }
  138     updtcompstatus(conflboxlst);
  139
  140     /* CONFLICT PROCESSING LOOP */
  141     do
  141     {
  142         ctrlpass(NOSELCT, CONFSEL);
  143
  144         hilite(CONFSTAT);
  145         hilite(CURSTAT);
  146         hilite(INSTRCT);
  147
  148         /* select material conflict from conflict list */
  149         conflbox = lboxinit(4, 37, 4, REVHIGHNORML, HELP, "", REVNORML, FRSINGLE,
  150             REVNORML, conflboxlst);
  151         dsplyconfstat(conflbox);
  152         CONFSELBOX :                    // conflict selection box
  153         while ((key = getkey()) != ESC && key != ENTR && key != F3)
  153         {
```

```
154             lboxctrl(conflbox, key);
155             dsplyconfstat(conflbox);
156         }
157         if (key == ESC)
157         {
158             if (verifyabort())
159                 goto CANCEL;
160             goto ABORTSUB;
161         }
162         else if (key == F3) goto CONFSELBOX;
163         choice = lboxsel(conflbox);
164         unhilite(CONFSTAT);
165         hilite(SELTCONF);
166         vratputf(CompWin, 11, 36, HELP, " %-38.38s ", choice);/* display conflict selection*/
167
168         ctrlpass(CONFSEL, NOSELCT);
169
170         if (isoverrideposs(conflbox))
171             acceptovrd = asktoacceptoverride();
172         else
173             acceptovrd = FALSE;
174         if (! acceptovrd)
174         {
175             ctrlpass(NOSELCT, SUPPSEL);
176
177             /* select substition material from AB supply */
178             LKUmodinit();
179             LKUmodposition(17, 3, 4);
180             LKUmodattributes(0x10 | 0x03, 0x20 | 0x00);
181             LKUmodselect(matsup_dbf, -1);
182             if ((selrecno = LKUmodlookup()) < 0)
182             {
183                 ctrlpass(SUPPSEL, NOSELCT);
184                 switch ((int) selrecno)
184                 {
185                 case (CANCEL_SEL) : if (verifyabort())
186                         goto CANCEL;
187                 case (ABORTSUB_SEL) : goto ABORTSUB;
188                 default :;
189                 }
190             }
191             else
191             {
192                 /* getsupprecstr and chkqtyvssupp require MATSUP.DBF */
193                 supprecstr = getsupprecstr(selrecno);
194                 hilite(SELTSUB);
195                 vratputf(CompWin, 13, 36, HELP, " %-38.38s ", supprecstr);/* display supply sel
ection*/
196
197                 ctrlpass(SUPPSEL, SELTQTY);
198
199                 acceptqty = FALSE;
200                 /* get qty, validate, verify substitution */
201                 do
201                 {
202                     acceptqty = FALSE;
203                     if ((subqty = getqty()) < 0)
203                     {
204                         switch ((int) subqty)
204                         {
205                         case (CANCEL_SEL) : if (verifyabort())
205                             {
206                                 ctrlpass(SELTQTY, NOSELCT);
207                                 goto CANCEL;
208                             }
209                             break;
210
211                         case (ABORTSUB_SEL) : ctrlpass(SELTQTY, NOSELCT);
212                             goto ABORTSUB;
213                         default :;
214                         }
215                     }
216                     else
216                     {
```

```
217                         validqty = chkqtyvssupp(subqty, selrecno);
218                         if (validqty)
219                             acceptqty = askifcorrectqty();
220                         else
221                             alerttoqtylimit(selrecno);
222                     }
223                 }
223                 while (! validqty || ! acceptqty);
224
225                 d4select(matsup_dbf);
226                 ctrlpass(SELTQTY, NOSELCT);
227                 dsplysubaccepted();
228             }
229             /* record substitution */
230             recordsub(conflbox, subqty, selrecno);
231         }
232         else
233             setovrdflag(conflbox);
234         rmfmlist(conflboxlst, choice);
235
236         ABORTSUB :
237         unhilite(SELTCONF);
238         unhilite(SELTSUB);
239         vratputf(CompWin, 11, 36, HELP, " %-38.38s ", " ");/* clr conf selection display*/
240         vratputf(CompWin, 13, 36, HELP, " %-38.38s ", " ");/* clr sub selection display */
241         lboxdel(conflbox);
242         if (! (listcnt(conflboxlst)))
243             success = TRUE;
244         updtcompstatus(conflboxlst);
245
246     }
246     while (! success);
247
248     if (success)
248     {
249         unhilite(INSTRCT);
250         dsplycompsuccessbox();
251     }
252
253     CANCEL :                            // cancel compromise
254     lboxdel(conflbox);
255     freelist(conflboxlst, 0);
256     free((REPDATANODE *) repinfoblk);
257     delcompwin();
258     delkeymapbar();
259
260     if (success)
261         return (SUCCESS);
262     return (FAILURE);
263 }
264
265 /* End of main ==========================================================*/
266
267
268 /* functions not called directly from DoCompromise() */
269 /*-----------------------------------------------------*/
270 char getavailstatofsel(LBOXSTA *);
271 char * bldlststr(REQLSTNODE *);
272 char * getconfstat(LBOXSTA *);
273 HWND dsplymessbox(void);
274 void delmessbox(HWND);
275 void delay(int);
276 /*-----------------------------------------------------*/
277
278 HWND initkeymapbar(void)
279 {
280  * ARGUMENT
281  *
282  * DESCRIPTION
283  *   Initializes key map bar for COMPROMISE environment
284  *
285  * RETURNS
286  *   (HWND) key map bar window handle
```

```
 287  *
 288  */
 289
 290     HWND keymapbar;
 291
 292 /* composed attributes */
 293     #define KEYS  REVERR
 294
 295     keymapbar = vcreat(3, 80, EMPHNORML, YES);
 296     vwind(keymapbar, 1, 80, 0, 0);
 297     vlocate(keymapbar, 24, 0);
 298     visible(keymapbar, YES, YES);
 299     vatps(0, 0, " F1 Help  / Select  DY Accept  ESC Cancel All");
 300     modattr(keymapbar, 0, 1, 2, KEYS);
 301     modattr(keymapbar, 0, 10, 1, KEYS);
 302     modattr(keymapbar, 0, 12, 1, KEYS);
 303     modattr(keymapbar, 0, 22, 3, KEYS);
 304     modattr(keymapbar, 0, 34, 3, KEYS);
 305     vatps(1, 0, " F1 Help  / Select  DY Accept  F2 Seek Matid  F3 Abort Sub  ESC Cancel All"
);
 306     modattr(keymapbar, 1, 1, 2, KEYS);
 307     modattr(keymapbar, 1, 10, 1, KEYS);
 308     modattr(keymapbar, 1, 12, 1, KEYS);
 309     modattr(keymapbar, 1, 22, 3, KEYS);
 310     modattr(keymapbar, 1, 34, 2, KEYS);
 311     modattr(keymapbar, 1, 49, 2, KEYS);
 312     modattr(keymapbar, 1, 63, 3, KEYS);
 313     vatps(2, 0, " F1 Help  <0..9,'.'> Enter Qty  BKSP Edit  DY Accept  F3 Abort Sub");
 314     modattr(keymaphar, 2, 1, 2, KEYS);
 315     modattr(keymapbar, 2, 11, 1, KEYS);
 316     modattr(keymapbar, 2, 14, 1, KEYS);
 317     modattr(keymapbar, 2, 17, 1, KEYS);
 318     modattr(keymapbar, 2, 32, 4, KEYS);
 319     modattr(keymapbar, 2, 43, 3, KEYS);
 320     modattr(keymapbar, 2, 55, 2, KEYS);
 321     visible(keymapbar, YES, NO);
 322     return (keymapbar);
 323 }
 324
 325 HWND initcompwin(void)
 325 {
 326 /*
 327  * ARGUMENT
 328  *
 329  * DESCRIPTION
 330  *   Initializes COMPROMISE environment window
 331  *
 332  * RETURNS
 333  *   (HWND) COMPROMISE environment window handle
 334  *
 335  */
 336
 337     HWND compwin;
 338
 339
 340     compwin = vcreat(22, 78, WHITE_ON_BLUE, YES);
 341     vlocate(compwin, 1, 1);
 342     vframe(compwin, WHITE_ON_BLUE, FRSINGLE);
 343     vmtitle(compwin, _TOP, CENTERJUST, WHITE_ON_BLUE, " COMPROMISE Environment ");
 344     visible(compwin, YES, YES);
 345     return (compwin);
 346 }
 347
 348 void delcompwin(void)
 348 {
 349 /*
 350  * ARGUMENT
 351  *
 352  * DESCRIPTION
 353  *   Deletes COMPROMISE environment window
 354  *
 355  * RETURNS
 356  *
 357  */
```

```
358     vdelete(CompWin, NONE);
359 }
360
361 void delkeymapbar(void)
361 {
362 /*
363  * ARGUMENT
364  *
365  * DESCRIPTION
366  *   Deletes key map bar window
367  *
368  * RETURNS
369  *
370  */
371     vdelete(KeymapBar, NONE);
372 }
373
374 void dsplycompsuccessbox(void)
374 {
375 /*
376  * ARGUMENT
377  *
378  * DESCRIPTION
379  *   Displays COMPROMISE "Success" message box
380  *
381  * RETURNS
382  *
383  */
384
385     HWND compsuccwin;
386
387     compsuccwin = dsplymessbox();
388     vratps(1, 0, GRYBTXT, "      **** SUCCESS ****       ");
389     vratps(3, 0, GRYBTXT, "   All conflicting material   ");
390     vratps(4, 0, GRYBTXT, "    requirements resolved!    ");
391     vratputf(compsuccwin, 7, 0, KEYALERT, "      < Press any key >       ");
392     getkey();
393     delmessbox(compsuccwin);
394     return;
395 }
396
397 HWND dsplymessbox(void)
397 {
398 /*
399  * ARGUMENT
400  *
401  * DESCRIPTION
402  *   Displays a standard message box
403  *
404  * RETURNS
405  *   (HWND) message box window handle
406  *
407  */
408
409     HWND messwin;
410
411     messwin = vcreat(8, 30, GREYBOX, YES);
412     vlocate(messwin, 8, 24);
413     vframe(messwin, GREYBOX, FRDOUBLE);
414     vshadow(messwin, CURRENT, SHADOW100, BOTTOMRIGHT);
415     visible(messwin, YES, YES);
416     return (messwin);
417 }
418
419 void delmessbox(HWND messwin)
419 {
420 /*
421  * ARGUMENT
422  *   (HWND) messwin - message box window handle
423  *
424  * DESCRIPTION
425  *   Deletes message box
426  *
427  * RETURNS
```

```
 428  *
 429  */
 430
 431     vdelete(messwin, NONE);
 432     vcurrent(CompWin);
 433     return;
 434 }
 435
 436 void dsplyrepinfo(REPDATANODE * blkptr)
 436 {
 437 /*
 438  * ARGUMENT
 439  *   (REPDATANODE *) blkptr - pointer to repair data block
 440  *
 441  * DESCRIPTION
 442  *   Displays the repair data block on the COMPROMISE environment window
 443  *
 444  * RETURNS
 445  *
 446  */
 447
 448     if (blkptr == (REPDATANODE *) NULL)
 448     {
 449         u4error(ERROR, "error", (char *) 0);
 450         return;
 451     }
 452     vratputf(CompWin, 1, 2, WHITE_ON_BLUE, "Repair information");
 453     vratputf(CompWin, 2, 2, ERR, " Repid       : %-15d", blkptr->repid);
 454     vratputf(CompWin, 3, 2, ERR, " Facnum      : B%-14d", blkptr->facnum);
 455     vratputf(CompWin, 4, 2, ERR, " Priority    : %-15d", blkptr->priority);
 456     vratputf(CompWin, 5, 2, ERR, " Elenum      : %-15d", blkptr->elenum);
 457     vratputf(CompWin, 6, 2, ERR, " Elem desc   : %-15s", blkptr->eledescstr);
 458     vratputf(CompWin, 7, 2, ERR, " Dam mode    : %-15s", blkptr->dammodestr);
 459     vratputf(CompWin, 8, 2, ERR, " Rep stratgy : %-15s", blkptr->repstgystr);
 460     return;
 461 }
 462
 463 void dsplycompstatus(int numofmatreqs)
 463 {
 464 /*
 465  * ARGUMENT
 466  *   (int) numofmatreqs - number of material requirements for a repair
 467  *
 468  * DESCRIPTION
 469  *   Displays COMPROMISE status block of repair (i.e. block specifies
 470  *   how many material conflicts exist out of total material requirements)
 471  *
 472  * RETURNS
 473  *
 474  */
 475
 476     vratputf(CompWin, 10, 2, WHITE_ON_BLUE, "Current status");
 477     vratputf(CompWin, 11, 2, REVHIGHHELP, " %2.2s ", " ");
 478     vratputf(CompWin, 11, 7, ERR, " conflicts out of      ");
 479     vratputf(CompWin, 12, 2, REVHIGHHELP, " %2d ", numofmatreqs);
 480     vratputf(CompWin, 12, 7, ERR, " material requirements ");
 481     return;
 482 }
 483
 484
 485
 486 char * bldlststr(REQLSTNODE * ptr)
 486 {
 487 /*
 488  * ARGUMENT
 489  *   (REQLSTNODE *) ptr - pointer to resource availability analysis list
 490  *                        node corresponding to material
 491  *                        requirement
 492  *
 493  * DESCRIPTION
 494  *    Builds a string representation of material record,
 495  *    string storage is within the same REQLSTNODE node
 496  *
 497  * RETURNS
```

```
498  *    (char *) Pointer to that string within the REQLSTNODE node to be
499  *    given to DATAWINDOWS listbox function
500  *
501  * DATABASES AFFECTED (whether direct or indirect)
502  *    matreq_dbf (no index used)
503  *
504  */
505
506     extern matreq_dbf;
507     char matdesc[60], unit[10];
508
509     d4select(matreq_dbf);
510     d4go(ptr->recno);
511     strcpy(matdesc, f4str(f4ref("MATDESC")));
512     strcpy(unit, f4str(f4ref("UNIT")));
513     strcpy(ptr->recstr, "");
514     sprintf(ptr->recstr, "%5ld %20.20s %7.1f %3.3s ",
515         f4long(f4ref("MATID")),
516         matdesc,
517         f4double(f4ref("QTY")),
518         unit);
519     return (ptr->recstr);
520 }
521
522 char * getfirstmatconf(REQLSTNODE * hdptr, REQLSTNODE **ptrptr)
522 {
523 /*
524  * ARGUMENT
525  *   (REQLSTNODE *) hdptr  - pointer to headnode of resource availability
526  *                           analysis list
527  *   (REQLSTNODE *) ptrptr - pointer to a pointer to the current material
528  *                           conflict node (allowing pointer modification)
529  *
530  * DESCRIPTION
531  *    Increments node pointer to first material conflict node in the
532  *    resource availability analysis list at which time a material
533  *    record node-resident text string is built for that conflict
534  *
535  * RETURNS
536  *    (char *) Pointer to that string within the REQLSTNODE node to be
537  *    given to DATAWINDOWS listbox function
538  *
539  * DATABASES AFFECTED (whether direct or indirect)
540  *    matreq_dbf (no index used)
541  *
542  */
543
544     char * retstrptr;
545
546     (* ptrptr) = (REQLSTNODE *) NULL;
547     if (hdptr == (REQLSTNODE *) NULL)
547     {
548         u4error(EMPTYLST, "Empty list error", (char *) 0);
549         return (NULL);
550     }
551     (* ptrptr) = (hdptr->next)->next;  /* skip past first eqp rec                    */
552
553     /* find first material record  */
554     for (; (* ptrptr)->type != 'M'; (* ptrptr) = (* ptrptr)->next)
555         if ((* ptrptr) == (REQLSTNODE *) NULL)
555         {
556             u4error(NONULLEXPT, "No null ptr expected", (char *) 0);
557             return (NULL);
558         }
559
560     /* find first material conflict */
561     for (; (* ptrptr)->availstatus == 'E'; (* ptrptr) = (* ptrptr)->next)
561     {
562         if ((* ptrptr) == (REQLSTNODE *) NULL)
562         {
563             u4error(NONULLEXPT, "No null ptr expected", (char *) 0);
564             return (NULL);
565         }
566     }
```

```
567     retstrptr = bldlststr(* ptrptr);
568     (* ptrptr) = (* ptrptr)->next;
569
570     return (retstrptr);
571 }
572
573
574 char * getnextmatconf(REQLSTNODE **ptrptr)
574 {
575 /*
576  * ARGUMENT
577  *   (REQLSTNODE *) ptrptr - pointer to a pointer to the current material
578  *                           conflict node (allowing pointer modification)
579  *
580  * DESCRIPTION
581  *    Increments node pointer to next material conflict node in the
582  *    resource availability analysis list at which time a material
583  *    record node-resident text string is built for that conflict
584  *
585  * RETURNS
586  *    (char *) Pointer to that string within the REQLSTNODE node to be
587  *    given to DATAWINDOWS listbox function
588  *
589  * DATABASES AFFECTED (whether direct or indirect)
590  *    matreq_dbf (no index used)
591  *
592  */
593
594     char * retstrptr;
595
596     if ((* ptrptr) == (REQLSTNODE *) NULL)
596     {
597         return (NULL);
598     }
599     /* find next material conflict */
600     for (; (* ptrptr)->availstatus == 'E'; (* ptrptr) = (* ptrptr)->next)
600     {
601         if (((* ptrptr)->next) == (REQLSTNODE *) NULL)
601         {
602             return (NULL);
603         }
604     }
605     retstrptr = bldlststr(* ptrptr);
606     (* ptrptr) = (* ptrptr)->next;
607
608     return (retstrptr);
609 }
610
611 char * getsupprecstr(long recno)
611 {
612 /*
613  * ARGUMENT
614  *   (long) recno - material supply record number (returned by LKU fn)
615  *
616  * DESCRIPTION
617  *   Takes supply record number and gets the specified record in string form
618  *
619  * RETURNS
620  *    (char *) string representation of material supply record
621  *
622  *
623  * DATABASES AFFECTED (whether direct or indirect)
624  *    matsup_dbf (no index used)
625  *
626  */
627
628     static char str[80];
629     MATSUPREC * msuprec;
630
631     msuprec = GetSuppRec(recno);
632     strcpy(str, "");
633     sprintf(str, "%5.5ld %20.20s %7.7s %3.3s ",
634         msuprec->matid,
635         msuprec->matdesc,
```

```
636         " ",
637         msuprec->unit);
638     free((MATSUPREC *) msuprec);
639     return (str);
640 }
641
642 char getavailstatofsel(LBOXSTA * lboxptr)
642 {
643 /*
644  * ARGUMENT
645  *   (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
646  *
647  * DESCRIPTION
648  *   Uses pointer in list box structure to access the availability status
649  *   code resident in the corresponding material resource requirement node
650  *   in the resource availability analysis list
651  *
652  * RETURNS
653  *   (char) availability status code (e.g. (M)issing,(I)nadequate,(O)verride)
654  *
655  */
656
657     return (* ((lboxptr->selectitem->listitem) + LBOXSTRLEN));
658
659 }
660
661 void setovrdflag(LBOXSTA * lboxptr)
661 {
662 /*
663  * ARGUMENT
664  *   (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
665  *
666  * DESCRIPTION
667  *   Uses pointer in list box structure to set override flag
668  *   resident in the corresponding material resource requirement node
669  *   in the resource availability analysis list
670  *
671  * RETURNS
672  *
673  */
674
675     ((REQLSTNODE *) ((lboxptr->selectitem->listitem) - sizeof (char) - sizeof (long)))->ovrd =
TRUE;
676 }
677
678 void recordsub(LBOXSTA * lboxptr  float qty, long recno)
678 {
679 /*
680  * ARGUMENT
681  *   (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
682  *   (float)         qty - substitution qty
683  *   (long)        recno - substitution record number
684  *
685  * DESCRIPTION
686  *   Uses pointer in list box structure to record the material substitution
687  *   using the appropriate fields (i.e. resrecno, resqty)
688  *   resident in the corresponding material resource requirement node
689  *   in the resource availability analysis list
690  *
691  * RETURNS
692  *
693  */
694
695     ((REQLSTNODE *) ((lboxptr->selectitem->listitem) - sizeof (char) - sizeof (long)))->resrecn
o = recno;
696     ((REQLSTNODE *) ((lboxptr->selectitem->listitem) - sizeof (char) - sizeof (long)))->resqty
= qty;
697     return;
698 }
699
700 char * getconfstat(LBOXSTA * lboxptr)
700 {
701 /*
702  * ARGUMENT
```

```
703  *    (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
704  *
705  * DESCRIPTION
706  *    Retrieves and decodes the conflict status of a material requirement
707  *
708  * RETURNS
709  *    (char *) conflict status explaination string
710  *
711  */
712
713     static char confstat[30];
714
715     switch (getavailstatofsel(lboxptr))
715     {
716     case ('I') : strcpy(confstat, "inadequate supply");
717         break;
718     case ('O') : strcpy(confstat, "priority override possible");
719         break;
720     case ('M') : strcpy(confstat, "mat not found in AB supply");
721         break;
722     default : strcpy(confstat, "invalid status");
723     }
724     return (confstat);
725 }
726
727 void dsplyconfstat(LBOXSTA * lboxptr)
727 {
728 /*
729  * ARGUMENT
730  *    (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
731  *
732  * DESCRIPTION
733  *    Displays the conflict status explaination string in the COMPROMISE
734  *    environment window
735  *
736  * RETURNS
737  *
738  */
739
740     vratputf(CompWin, 8, 46, HELP, " %-26.26s ", getconfstat(lboxptr));
741
742 }
743
744 int isoverrideposs(LBOXSTA * lboxptr)
744 {
745 /*
746  * ARGUMENT
747  *    (LBOXSTA *) lboxptr - pointer to list box structure of conflict list
748  *
749  * DESCRIPTION
750  *    Checks to see if a priority override has been designated for the
751  *    selected material conflict
752  *
753  * RETURNS
754  *    (int) TRUE or FALSE
755  *
756  */
757
758     if (getavailstatofsel(lboxptr) == 'O')
759         return (TRUE);
760     else
761         return (FALSE);
762 }
763
764
765 void setupwinformat(void)
765 {
766 /*
767  * ARGUMENT
768  *
769  * DESCRIPTION
770  *    Sets up the COMPROMISE environment window format
771  *
772  * RETURNS
```

```
 773  *
 774  */
 775
 776     int i;
 777
 778     vatps(1, 35, "   Conflicting mat requirements ");
 779     vatps(15, 2, "   AB material supply ");
 780     vatps(7, 37, "CONFLICT ");
 781     vatps(8, 37, "STATUS : ");
 782     vratputf(CompWin, 8, 47, HELP, " %-26.26s ", " ");
 783     /* setup selection display section */
 784     vatps(10, 36, "Selected conflict");
 785     vratputf(CompWin, 11, 36, HELP, " %-38.38s ", " ");
 786     vatps(12, 36, "Selected substitution");
 787     vratputf(CompWin, 13, 36, HELP, " %-38.38s ", " ");
 788     vatps(15, 60, "INSTRUCTIONS: ");
 789     for (i = 0; i <= 5; i++)
 790         vatpas(16 + i, 60, HELP, "                ");
 791     return;
 792 }
 793
 794
 795
 796 void hilite(int item)
 796 {
 797 /*
 798  * ARGUMENT
 799  *    (int) item - mneumonic constant corresponding to a COMPROMISE
 800  *                 environment window item
 801  *
 802  * DESCRIPTION
 803  *   Hilites the specified item on the screen
 804  *
 805  * RETURNS
 806  *
 807  */
 808     switch (item)
 808     {
 809     case (CONFSTAT) :
 810         /* hilite "CONFLICT" , hilite "STATUS :" */
 811         modattr(CompWin, 7, 37, 8, HILITE);
 812         modattr(CompWin, 8, 37, 8, HILITE);
 813         break;
 814     case (SELTCONF) :
 815         /* hilite "Selected conflict"   */
 816         modattr(CompWin, 10, 35, 18, HILITE);
 817         break;
 818     case (SELTSUB) :
 819         /* hilite "Selected substitution" */
 820         modattr(CompWin, 12, 36, 21, HILITE);
 821         break;
 822     case (CURSTAT) :
 823         /* hilite "Current status" */
 824         modattr(CompWin, 10, 2, 14, HILITE);
 825         break;
 826     case (INSTRCT) :
 827         /* hilite "Instruction:" */
 828         modattr(CompWin, 15, 60, 14, HILITE);
 829         break;
 830     default :;
 831     }
 832     return;
 833 }
 834
 835
 836
 837 void unhilite(int item)
 837 {
 838 /*
 839  * ARGUMENT
 840  *    (int) item - mneumonic constant corresponding to a COMPROMISE
 841  *                 environment window item
 842  *
 843  * DESCRIPTION
```

```
844  *   Unhilites the specified item on the screen
845  *
846  * RETURNS
847  *
848  */
849     switch (item)
849     {
850     case (CONFSTAT) :
851         /* unhilite "CONFLICT" , hilite "STATUS :" */
852         modattr(CompWin, 7, 37, 8, CURRENT);
853         modattr(CompWin, 8, 37, 8, CURRENT);
854         break;
855     case (SELTCONF) :
856         /* unhilite "Selected conflict"   */
857         modattr(CompWin, 10, 35, 18, CURRENT);
858         break;
859     case (SELTSUB) :
860         /* unhilite "Selected substitution" */
861         modattr(CompWin, 12, 36, 21, CURRENT);
862         break;
863     case (CURSTAT) :
864         /* unhilite "Current status" */
865         modattr(CompWin, 10, 2, 14, CURRENT);
866         break;
867     case (INSTRCT) :
868         /* unhilite "Instruction:" */
869         modattr(CompWin, 15, 60, 14, CURRENT);
870         break;
871     default :;
872     }
873     return;
874 }
875
876
877
878 void ctrlpass(int from, int to)
878 {
879 /*
880  * ARGUMENT
881  *    (int) from - current control point
882  *    (int) to   - next control point
883  *
884  * DESCRIPTION
885  *   Coordinates screen control of the COMPROMISE envrionmant between
886  *   the different control points
887  *
888  * RETURNS
889  *
890  */
891
892     switch (from)
892     {
893     case (CONFSEL) :
894         /* unselect "Conflicting mat requirements" */
895         vatputs(CompWin, 1, 35, "   ");
896         modattr(CompWin, 1, 35, 32, CURRENT);
897         break;
898
899     case (SUPPSEL) :
900         /* unselect "AB material supply" */
901         vatputs(CompWin, 15, 2, "   ");
902         modattr(CompWin, 15, 4, 20, CURRENT);
903         break;
904     case (SELTQTY) :
905         /* unselect "Selected substitution" */
906         vatputs(CompWin, 13, 33, "  ");
907         modattr(CompWin, 12, 36, 21, CURRENT);
908         break;
909     case (NOSELCT) :
910     default :;
911     }
912     switch (to)
912     {
913     case (CONFSEL) :
```

```
914         /* select "Conflicting material requirements" */
915         vratputf(CompWin, 1, 35, SELBLINK, " ");
916         modattr(CompWin, 1, 37, 30, SELNORM);
917         vloc(KeymapBar, 0, 0);
918         vratputs(CompWin, 16, 60, HELP, "                ");
919         vratputs(CompWin, 17, 60, HELP, " > Select a     ");
920         vratputs(CompWin, 18, 60, HELP, "   material     ");
921         vratputs(CompWin, 19, 60, HELP, "   conflict to  ");
922         vratputs(CompWin, 20, 60, HELP, "   resolve      ");
923         vratputs(CompWin, 21, 60, HELP, "                ");
924         break;
925     case (SUPPSEL) :
926         /* select "AB material supply" */
927         vratputf(CompWin, 15, 2, SELBLINK, " ");
928         modattr(CompWin, 15, 4, 20, SELNORM);
929         vloc(KeymapBar, 1, 0);
930         vratputs(CompWin, 16, 60, HELP, "                ");
931         vratputs(CompWin, 17, 60, HELP, " > Select a     ");
932         vratputs(CompWin, 18, 60, HELP, "   substitute   ");
933         vratputs(CompWin, 19, 60, HELP, "   material     ");
934         vratputs(CompWin, 20, 60, HELP, "                ");
935         vratputs(CompWin, 21, 60, HELP, "                ");
936         break;
937     case (SELTQTY) :
938         /* unselect "Selected substitution" */
939         vratputf(CompWin, 13, 33, SELBLINK, " ");
940         vloc(KeymapBar, 2, 0);
941         vratputs(CompWin, 16, 60, HELP, "                ");
942         vratputs(CompWin, 17, 60, HELP, " > Select qty   ");
943         vratputs(CompWin, 18, 60, HELP, "   to           ");
944         vratputs(CompWin, 19, 60, HELP, "   substitute   ");
945         vratputs(CompWin, 20, 60, HELP, "                ");
946         vratputs(CompWin, 21, 60, HELP, "                ");
947         break;
948     case (NOSELCT) :
949         vratputs(CompWin, 16, 60, HELP, "                ");
950         vratputs(CompWin, 17, 60, HELP, "                ");
951         vratputs(CompWin, 18, 60, HELP, "                ");
952         vratputs(CompWin, 19, 60, HELP, "                ");
953         vratputs(CompWin, 20, 60, HELP, "                ");
954         vratputs(CompWin, 21, 60, HELP, "                ");
955     default :;
956     }
957     return;
958 }
959
960 void updtcompstatus(LISTITEM * conflboxlst)
960 {
961 /*
962  * ARGUMENT
963  *   (LISTITEM *) conflboxlst - pointer to conflict list box list
964  *
965  * DESCRIPTION
966  *   Updates the COMPROMISE status by checking the current number of
967  *   material conflicts remaining and displaying the new status on the
968  *   COMPROMISE environment window
969  *
970  * RETURNS
971  *
972  */
973
974     vratputf(CompWin, 11, 2, REVHIGHHELP, "  %d ", listcnt(conflboxlst));
975
976 }
977
978 float getqty(void)
978 {
979 /*
980  * ARGUMENT
981  *
982  * DESCRIPTION
983  *   Gets the substitution qty from the user
984  *
985  * RETURNS
```

```
 986  *    (float) n          - qty selected
 987  *            CANCEL_SEL - if ESC hit
 988  *            ABORT_SEL  - if F3 hit
 989  *
 990  */
 991
 992     unsigned int key;
 993     int decpntfound, cnt, digafterdecpnt, done;
 994     char str[10];
 995
 996     vratputf(CompWin, 13, 64, SELQTYBLINK, "%7.7s", "   ?   ");/* prompt for qty        */
 997     for (cnt = 0,
 998             decpntfound = FALSE,
 999             digafterdecpnt = 0,
1000             done = FALSE,
1001             str[0] = '\0',
1002             key = getkey();
1003             (! done);)
1003     {
1004         if (cnt == 0)
1005             vratputf(CompWin, 13, 64, SELQTY, "%7.7s", "      ");/* clear qty space     */
1006         switch (key)
1006         {
1007         case (ESC) : return ((float) CANCEL_SEL);/* ESC hit (cancel compromise)         */
1008         case (F3) : return ((float) ABORTSUB_SEL);/* F3 hit (abort substitution)        */
1009         case (BKSP) : if (cnt > 0)
1009             {
1010                 cnt--;
1011                 str[cnt] = '\0';
1012                 vratputf(CompWin, 13, 64 + cnt, SELQTY, "%c", ' ');
1013                 if (decpntfound && (digafterdecpnt == 1))
1014                     digafterdecpnt = 0;
1015                 else if (decpntfound && (digafterdecpnt == 0))
1016                     decpntfound = FALSE;
1017             }
1018             break;
1019         case (ENTR) : if (str[0] != '\0')
1019             {
1020                 done = TRUE;
1021                 dmpxtrakeys();
1022             }
1023             else
1024                 /* prompt for qty again */
1025                 vratputf(CompWin, 13, 64, SELQTYBLINK, "%7.7s", "   ?   ");
1026             break;
1027         case ('0') :
1028         case ('1') :
1029         case ('2') :
1030         case ('3') :
1031         case ('4') :
1032         case ('5') :
1033         case ('6') :
1034         case ('7') :
1035         case ('8') :
1036         case ('9') : if ((cnt <= 6) && (digafterdecpnt != 1))
1036             {
1037                 str[cnt] = key;
1038                 str[cnt + 1] = '\0';
1039                 vratputf(CompWin, 13, 64 + cnt, SELQTY, "%c", key);
1040                 cnt++;
1041                 if (decpntfound)
1042                     digafterdecpnt = 1;
1043             }
1044             else
1045                 vbeep();
1046             break;
1047         case ('.') : if ((cnt <= 6) && (! decpntfound))
1047             {
1048                 decpntfound = TRUE;
1049                 str[cnt] = key;
1050                 str[cnt + 1] = '\0';
1051                 vratputf(CompWin, 13, 64 + cnt, SELQTY, "%c", key);
1052                 cnt++;
1053             }
```

```
1054             else
1055                 vbeep();
1056             break;
1057         default : vbeep();              /* INVALID keystroke                */
1058         }
1059         if (! done)
1060             key = getkey();
1061     }
1062     vratputf(CompWin, 13, 64, HELP, "%7.7s", str);
1063
1064     return (atof(str));
1065 }
1066
1067 int asktoacceptoverride(void)
1067 {
1068 /*
1069  * ARGUMENT
1070  *
1071  *
1072  * DESCRIPTION
1073  *   Asks the user if override should be accepted
1074  *
1075  * RETURNS
1076  *
1077  *
1078  * DATABASES AFFECTED (whether direct or indirect)
1079  *
1080  */
1081
1082     HWND win;
1083     unsigned int key;
1084
1085     win = dsplymessbox();
1086     vratputs(win, 1, 0, GRYBTXT, " **** PRIORITY  OVERRIDE **** ");
1087     vratputs(win, 3, 0, GRYBTXT, "           possible!          ");
1088     vratputs(win, 7, 0, KEYMESS, "   Accept override (Y):  ?    ");
1089     modattr(win, 7, 24, 3, FLSHCHR);
1090     while (1)
1090     {
1091         key = getkey();
1092         switch (key)
1092         {
1093         case ('Y') :
1094         case ('y') :
1095         case (ENTR) : vratputs(win, 7, 24, HELP, " Y ");
1096             delay(2);
1097             delmessbox(win);
1098             dmpxtrakeys();
1099             return (TRUE);
1100         case ('N') :
1101         case ('n') : vratputs(win, 7, 24, HELP, " N ");
1102             delay(2);
1103             delmessbox(win);
1104             dmpxtrakeys();
1105             return (FALSE);
1106         default : vbeep();              /* INVALID keystroke                */
1107         }
1108     }
1109 }
1110
1111 void delay(int sec)
1111 {
1112 /*
1113  * ARGUMENT
1114  *
1115  *
1116  * DESCRIPTION
1117  *   Implements a delay with 1:1 second correspondence on 25MHz 386
1118  *
1119  * RETURNS
1120  *
1121  *
1122  * DATABASES AFFECTED (whether direct or indirect)
1123  *
```

```
1124  */
1125
1126      long cnt, time;
1127
1128      time = sec;
1129      for (time = 0; time < sec; time++)
1129      {
1130          for (cnt = 300000L; cnt > 0; cnt--);
1131      }
1132  }
1133
1134  int verifyabort(void)
1134  {
1135  /*
1136   * ARGUMENT
1137   *
1138   *
1139   * DESCRIPTION
1140   *   Asks user to verify abort selection
1141   *
1142   * RETURNS
1143   *
1144   *
1145   * DATABASES AFFECTED (whether direct or indirect)
1146   *
1147   */
1148
1149      HWND win;
1150      unsigned int key;
1151
1152      win = dsplymessbox();
1153      vratputs(win, 0, 0, GRYBTXT, "                                ");
1154
1155      vratputs(win, 1, 0, GRYBTXT, "  **  CANCEL  COMPROMISE  **  ");
1156      modattr(win, 1, 2, 2, SELQTYBLINK);
1157      modattr(win, 1, 26, 2, SELQTYBLINK);
1158      vratputs(win, 2, 0, GRYBTXT, "                                ");
1159      vratputs(win, 3, 0, GRYBTXT, "              chosen!           ");
1160      vratputs(win, 4, 0, GRYBTXT, "                                ");
1161      vratputs(win, 7, 0, KEYMESS, "     Are you sure (N):  ?       ");
1162      modattr(win, 7, 23, 3, FLSHCHR);
1163      while (1)
1163      {
1164          key = getkey();
1165          switch (key)
1165          {
1166          case ('N') :
1167          case ('n') :
1168          case (ENTR) : vratputs(win, 7, 23, HELP, " N ");
1169              delay(2);
1170              vdelete(win, NONE);
1171              vcurrent(CompWin);
1172              dmpxtrakeys();
1173              return (FALSE);
1174          case ('Y') :
1175          case ('y') : vratputs(win, 7, 23, HELP, " Y ");
1176              delay(2);
1177              vdelete(win, NONE);
1178              vcurrent(CompWin);
1179              dmpxtrakeys();
1180              return (TRUE);
1181          default : vbeep();              /* INVALID keystroke                    */
1182          }
1183      }
1184  }
1185
1186  int chkqtyvssupp(float qty, long recno)
1186  {
1187  /*
1188   * ARGUMENT
1189   *
1190   *
1191   * DESCRIPTION
1192   *   Checks to makes sure qty entered for substitution is not
```

```
1193  *   more than is available in supply
1194  *
1195  * RETURNS
1196  *
1197  *
1198  * DATABASES AFFECTED (whether direct or indirect)
1199  *
1200  */
1201
1202     MATSUPREC * msuprec;
1203
1204     msuprec = GetSuppRec(recno);
1205     if (msuprec->qty >= qty)
1205     {
1206         free((MATSUPREC *) msuprec);
1207         return (TRUE);
1208     }
1209     free((MATSUPREC *) msuprec);
1210     return (FALSE);
1211 }
1212
1213 int askifcorrectqty(void)
1213 {
1214 /*
1215  * ARGUMENT
1216  *
1217  *
1218  * DESCRIPTION
1219  *   Asks user if qty entered is correct
1220  *
1221  * RETURNS
1222  *
1223  *
1224  * DATABASES AFFECTED (whether direct or indirect)
1225  *
1226  */
1227
1228     HWND win;
1229     unsigned int key;
1230
1231     win = dsplymessbox();
1232     vratputs(win, 2, 0, GRYBTXT, "  IS THIS QUANTITY CORRECT ?  ");
1233     vratputs(win, 6, 0, KEYMESS, "    Please verify (Y):  ?     ");
1234     modattr(win, 6, 23, 3, FLSHCHR);
1235     while (1)
1235     {
1236         key = getkey();
1237         switch (key)
1237         {
1238         case ('Y') :
1239         case ('y') :
1240         case (ENTR) : vratputs(win, 6, 23, HELP, " Y ");
1241             delay(1);
1242             vdelete(win, NONE);
1243             vcurrent(CompWin);
1244             dmpxtrakeys();
1245             return (TRUE);
1246         case ('N') :
1247         case ('n') : vratputs(win, 6, 23, HELP, " N ");
1248             delay(1);
1249             vdelete(win, NONE);
1250             vcurrent(CompWin);
1251             dmpxtrakeys();
1252             return (FALSE);
1253         default : vbeep();              /* INVALID keystroke                    */
1254         }
1255     }
1256 }
1257
1258 void dsplysubaccepted(void)
1258 {
1259 /*
1260  * ARGUMENT
1261  *
```

```
1262  *
1263  * DESCRIPTION
1264  *   Displays "substitution accepted" box
1265  *
1266  * RETURNS
1267  *
1268  *
1269  * DATABASES AFFECTED (whether direct or indirect)
1270  *
1271  */
1272
1273     HWND win;
1274
1275     win = dsplymessbox();
1276     vratputs(win, 1, 0, GRYBTXT, "  ****     MATERIAL     ****  ");
1277     vratputs(win, 3, 0, GRYBTXT, "         SUBSTITUTION          ");
1278     vratputs(win, 6, 0, KEYALERT, "           ACCEPTED            ");
1279     delay(3);
1280     delmessbox(win);
1281     vcurrent(CompWin);
1282 }
1283
1284 void alerttoqtylimit(long recno)
1284 {
1285 /*
1286  * ARGUMENT
1287  *
1288  *
1289  * DESCRIPTION
1290  *   Displays supply qty limit
1291  *
1292  * RETURNS
1293  *
1294  *
1295  * DATABASES AFFECTED (whether direct or indirect)
1296  *
1297  */
1298
1299     HWND win;
1300     MATSUPREC * msuprec;
1301
1302     msuprec = GetSuppRec(recno);
1303     win = dsplymessbox();
1304     vratputs(win, 1, 0, GRYBTXT, " ****   I N V A L I D    **** ");
1305     vratputs(win, 3, 0, GRYBTXT, "       QUANTITY  ENTRY         ");
1306     vratputf(win, 5, 0, GRYBTXT, " Total qty avail: %7.1f %3.3s ", msuprec->qty, msuprec->unit)
;
1307     modattr(win, 5, 18, 11, KEYMESS);
1308     vratputs(win, 7, 0, KEYALERT, "      < Press any key >       ");
1309     getkey();
1310     delay(1);
1311     dmpxtrakeys();
1312     delmessbox(win);
1313     free((MATSUPREC *) msuprec);
1314     vcurrent(CompWin);
1315     return;
1316 }
1317
```

```
 1 /*
 2  * dbutil.c
 3  *
 4  *     utility functions access repair info in dbases
 5  */
 6
 7 #include <stdlib.h>
 8 #include <string.h>
 9 #include <stdio.h>
10
11 #include <d4base.h>
12 #include <dw.h>
13
14 #include "rmrsutil.h"
15 #include "sysdefs.h"
16 #include "rmrserr.h"
17 #include "miscutil.h"
18
19
20 REPDATANODE * GetRepData(long repid)
20 {
21 /*
22  * ARGUMENT
23  *   (long) repid - repair id
24  *
25  * DESCRIPTION
26  *   Retrieves repair data from databases for use in repair data block
27  *
28  * RETURNS
29  *   (REPDATANODE *) pointer to structure containing the information
30  *   needed for the repair data block as used in the COMPROMISE mode
31  *
32  * DATABASES AFFECTED (whether direct or indirect)
33  *    repair_dbf (repair_ndx1 - REPID index       ) - d4seek used
34  *    repinfo_dbf (repinfo_ndx1 - REPID index     ) - d4seek used
35  *
36  */
37
38     REPDATANODE * buffer;
39
40     buffer = (REPDATANODE *) malloc(sizeof (REPDATANODE));
41     d4select(repair_dbf);
42     i4select(repair_ndx1);
43     d4seek_double(repid);
44     buffer->repid = repid;
45     buffer->facnum = f4long(f4ref("FACNUM"));
46     buffer->priority = f4int(f4ref("PRIORITY"));
47
48     repinfo_dbf = d4use_excl("REPINFO.DBF");
49     d4select(repinfo_dbf);
50     repinfo_ndx1 = i4open("REPINDX1.NDX");
51     i4select(repinfo_ndx1);
52     d4seek_double(repid);
53     buffer->elenum = f4int(f4ref("ELENUM"));
54     strcpy(buffer->eledescstr, f4str(f4ref("ELEDESC")));
55     strcpy(buffer->dammodestr, f4str(f4ref("DAMMODE")));
56     strcpy(buffer->repstgystr, f4str(f4ref("REPSTGY")));
57     trimstr(buffer->eledescstr);
58     trimstr(buffer->dammodestr);
59     trimstr(buffer->repstgystr);
60
61     d4select(repinfo_dbf);
62     d4close();
63     return (buffer);
64 }
65
66 MATSUPREC * GetSuppRec(long recno)
66 {
67 /*
68  * ARGUMENT
69  *   (long) recno - record number of desired material supply record
70  *
71  * DESCRIPTION
72  *   Retrieves entire specified material supply record from material
```

```
 73  *   supply database
 74  *
 75  * RETURNS
 76  *   (MATSUPREC *) pointer to structure containing the information
 77  *   from the specified material supply record
 78  *
 79  * DATABASES AFFECTED (whether direct or indirect)
 80  *    matsup_dbf (no index used)
 81  *
 82  */
 83
 84     MATSUPREC * buffer;
 85
 86     d4select(matsup_dbf);
 87     if (d4go(recno))
 87     {
 88         u4error(RECACESERR, "Record access error", (char *) 0);
 89         return ((MATSUPREC *) NULL);
 90     }
 91     buffer = (MATSUPREC *) malloc(sizeof (MATSUPREC));
 92
 93     buffer->matid = f4long(f4ref("MATID"));
 94     strcpy(buffer->matdesc, f4str(f4ref("MATDESC")));
 95     buffer->qty = f4double(f4ref("QTY"));
 96     strcpy(buffer->unit, f4str(f4ref("UNIT")));
 97     return (buffer);
 98 }
 99
100 int GetFacPrty(long facnum)
100 {
101 /*
102  * ARGUMENT
103  *   (long) facnum - facility number designation
104  *
105  * DESCRIPTION
106  *   Retrieves AB assigned facility priority from AB specific facility
107  *   priority database
108  *
109  * RETURNS
110  *   (int)    n - facility priority
111  *            0 - if facility listing not found
112  *
113  * DATABASES AFFECTED (whether direct or indirect)
114  *    facprt_dbf (facprt_ndx1)
115  *
116  */
117
118     int status, prty;
119
120     /* fetch associated facility priority from FACPRTY.DBF */
121     open_facprt(0);
122     d4select(facprt_dbf);
123     i4select(facprt_ndx1);
124     status = d4seek_double(facnum);
125     prty = f4int(f4ref("PRIORITY"));
126     d4close();
127     if (! status)
128         return (prty);
129     else
130         return (0);
131 }
```

```
 1 /*
 2  * itemfns.cpp
 3  *
 4  *     This file contains the functions that are called from the
 5  *     menu bar pull-down selections. These functions are listed
 6  *     below beside their menu selection:
 7  *
 8  *       System     > About ...              : SystAbotFn()
 9  *                    Proc info ...          : SystInfoFn()
10  *                    DOS shell              : SystXDOSFn()
11  *                    Comm Prog              : SystCOMMFn()
12  *                    Quit                   : SystQuitFn()
13  *       Init/Setup > Change def specs       : InitDefsFn()
14  *                    Load new supply file > AB mat supply file   : InitMatlFn()
15  *                                           AB equip supply file : InitEquiFn()
16  *       Process    > Auto Mode              : ProcAutoFn()
17  *                    Single facility        : ProcFaclFn()
18  *                    Compromise Retry       : ProcCompFn()
19  *                    Cancel repair          : ProcCancFn()
20  *       Schedule   > Auto Mode              : SchdAutoFn()
21  *                    Upgrade priority       : SchdUpgdFn()
22  *                    Set order              : SchdOrdrFn()
23  *    ChangeStatus  > Completed              : ChngCompFn()
24  *                    Canceled               : ChngCancFn()
25  *                    Restore                : ChngRestFn()
26  *                    Delete All             : ChngDelAFn()
27  *       Gen Report > Repair Schedule       >  by repair team : GnRpSchdTeamFn()
28  *                                             by facility    : GnRpSchdFaclFn()
29  *                    Completed repairs      : GnRpCompFn()
30  *                    Canceled repairs       : GnRpCancFn()
31  *                    Suspended repairs      : GnRpSuspFn()
32  *                    Possible repairs       : GnRpPossFn()
33  *                    Allocated materials    : GnRpAMatFn()
34  *                    Used materials         : GnRpUMatFn()
35  *                    AB materials supply :  GnRpABMtFn()
36  *                    AB equipment supply :  GnRpABEqFn()
37  *
38  *
39 */
40
41 #include <stdio.h>
42 #include <dir.h>
43 #include <stdlib.h>
44 #include <string.h>
45 #include "dw.h"
46 #include "dwmenu.h"
47 #include "dwsystem.h"
48 #include "d4base.h"
49
50 #include "rmrs.h"
51
52 extern void IntroScreen();
53
54 HWND win;
55
56 #define ERROREND -1
57
58 int SystAbotFn(void)
59 {
60     IntroScreen();
61     return (0);
62 }
63
64 int SystInfoFn(void)
65 {
66     long i;
67     HWND InfoFnWin;
68     unsigned short int * screenbucket = NULL;
69     InfoFnWin = vcreat(7, 40, ERR, YES);
70     d4close_all();
71     vlocate(InfoFnWin, 9, 20);
72     vframe(InfoFnWin, ERR, FRDOUBLE);
73     vshadow(InfoFnWin, CURRENT, SHADOW100, BOTTOMRIGHT);
74     vmtitle(InfoFnWin, _TOP, CENTERJUST, ERR, " RMRS Processing Information ");
```

```
 75     visible(InfoFnWin, YES, YES);
 76     for (i = 0; i < 3200000; i++);
 77
 78     vdelete(InfoFnWin, NONE);
 79     return (0);
 80 }
 81
 82 int SystXDOSFn(void)
 83 {
 84     int curpos;
 85     HWND win;
 86     int stat;
 87
 88     win = vcreat(21, 78, REVNORML, YES);
 89     vlocate(win, 2, 1);
 90     vframe(win, REVHIGHNORML, FRSINGLE);
 91     vtitle(win, REVHIGHNORML, "< DOS Window - Type EXIT to quit >");
 92     visible(win, YES, YES);
 93
 94
 95     stat = vredir(ON, win);
 96     if (stat < DWSUCCESS)
 97     {
 98         printf("Redirection Failure\n");
 99         vdelete(win, NONE);
100     }
101     else
102     {
103         vratps(1, 1, NORML, " Type EXIT to return to RMRS ... ");
104         curdrag(win, ON);
105         vredir(ON, win);
106         system("command");
107     }
108     vredir(OFF, win);
109     vredir(0, 0);
110     vclear(win);
111     vdelete(win, NONE);
112
113     return (0);
114 }
115
116 int SystQuitFn(void)
117 {
118     HWND QuitFnWin;
119     char reply;
120
121     QuitFnWin = vcreat(7, 40, EMPHNORML, YES);
122     vlocate(QuitFnWin, 9, 20);
123     vframe(QuitFnWin, EMPHNORML, FRDOUBLE);
124     vshadow(QuitFnWin, CURRENT, SHADOW75, BOTTOMRIGHT);
125     visible(QuitFnWin, YES, YES);
126     vatputs(QuitFnWin, 3, 2, "Are you sure you want to quit (Y/N)?");
127     if (((reply = getkey()) == 'Y') ||(reply == 'y'))
128     {
129         d4init_undo();
130         fcloseall();
131         /* reset default video mode */
132         vexit(0);
133     }
134     vdelete(QuitFnWin, NONE);
135     return (0);
136 }
137
138 int InitMatlFn(void)
139 {
140     long i;
141     HWND UserFnWin2;
142
143     UserFnWin2 = vcreat(7, 40, ERR, YES);
144     vlocate(UserFnWin2, 9, 20);
145     vframe(UserFnWin2, ERR, FRDOUBLE);
146     vshadow(UserFnWin2, CURRENT, SHADOW100, BOTTOMRIGHT);
147     vmtitle(UserFnWin2, _TOP, CENTERJUST, ERR, " none ");
148     visible(UserFnWin2, YES, YES);
```

```
149     for (i = 0; i < 3200000; i++);
150
151
152     vdelete(UserFnWin2, NONE);
153     return (0);
154 }
155
156 int InitEquiFn(void)
157 {
158     long i;
159     HWND UserFnWin3;
160
161     UserFnWin3 = vcreat(7, 40, ERR, YES);
162     vlocate(UserFnWin3, 9, 20);
163     vframe(UserFnWin3, ERR, FRDOUBLE);
164     vshadow(UserFnWin3, CURRENT, SHADOW100, BOTTOMRIGHT);
165     vmtitle(UserFnWin3, _TOP, CENTERJUST, ERR, " none ");
166     visible(UserFnWin3, YES, YES);
167     for (i = 0; i < 3200000; i++);
168
169
170     vdelete(UserFnWin3, NONE);
171     return (0);
172 }
173
174
175 int ProcAutoFn(void)
176 {
177     int DoProcessAuto();
178     long i;
179     short x;
180     char str[6];
181     extern HWND F1HelpWin;
182     int repair_ref;
183     int done;
184     struct ffblk file;
185
186     win = vcreat(7, 50, EMPHNORML, YES);
187     vlocate(win, 9, 20);
188     vframe(win, EMPHNORML, FRDOUBLE);
189     vshadow(win, CURRENT, SHADOW75, BOTTOMRIGHT);
190     vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Auto-mode Repair Processing ");
191     visible(win, YES, YES);
192
193     DoProcessAuto();
194
195     vdelete(win, NONE);
196     return (0);
197 }
198
199 int ProcFaclFn(void)
200 {
201     char * srchfor = "*.EQP";
202     MENUHDR * PDESDirHdr;
203     LISTITEM * file, * temp;
204     HWND win;
205     int num, row = 3, col = 3;
206
207     win = vopen(18, 60, REVNORML, REVNORML, FRDOUBLE, "< PDES files remaining to be processed >
");
208     vatputs(win, 2, 3, "Select facility");
209     vatputs(win, 3, 3, " to process ->>");
210     vatputs(win, 16, 3, "Ctrl-Enter to end selection.");
211     pclrattr(NORML);
212     pclrchar((char) 0xb0);
213     sclear();
214     PDESDirHdr = MNUCreateHdr(POPUP);
215     PDESDirHdr->toprow = 3;
216     PDESDirHdr->topcol = 30;
217     PDESDirHdr->maxrows = 9;
218     num = MNUSelectFiles(PDESDirHdr, srchfor, & file);
219     vclear(win);
220     vatputf(win, 1, 3, "%d items selected", num);
221     if (num)
```

```
221     {
222         temp = file;
223         do
223         {
224             vatputs(win, row, col, temp->listitem);
225             if (row++ > 17)
225             {
226                 row = 3;
227                 col += 15;
228             }
229             temp = temp->lnext;
230         }
230         while (temp != file);
231     }
232     vatputs(win, 15, 3, "Hit any key to exit to main menu. ");
233     getkey();
234     MNUDeleteMenu(PDESDirHdr);
235     vdelete(win, NONE);
236     return (1);
237 }
238
239
240 int ProcCompFn(void)
241 {
242     long i;
243     HWND UserFnWin6;
244
245     UserFnWin6 = vcreat(7, 40, ERR, YES);
246     vlocate(UserFnWin6, 9, 20);
247     vframe(UserFnWin6, ERR, FRDOUBLE);
248     vshadow(UserFnWin6, CURRENT, SHADOW100, BOTTOMRIGHT);
249     vmtitle(UserFnWin6, _TOP, CENTERJUST, ERR, " none ");
250     visible(UserFnWin6, YES, YES);
251     for (i = 0; i < 3200000; i++);
252
253
254     vdelete(UserFnWin6, NONE);
255     return (0);
256 }
257
258 int ProcCancFn(void)
259 {
260     long i;
261     HWND UserFnWin7;
262
263     UserFnWin7 = vcreat(7, 40, ERR, YES);
264     vlocate(UserFnWin7, 9, 20);
265     vframe(UserFnWin7, ERR, FRDOUBLE);
266     vshadow(UserFnWin7, CURRENT, SHADOW100, BOTTOMRIGHT);
267     vmtitle(UserFnWin7, _TOP, CENTERJUST, ERR, " none ");
268     visible(UserFnWin7, YES, YES);
269     for (i = 0; i < 3200000; i++);
270
271
272     vdelete(UserFnWin7, NONE);
273     return (0);
274 }
275
276 int SchdFullFn(void)
277 {
278     SchedAuto(1);
279     return (0);
280 }
281
282 int SchdIncFn(void)
283 {
284     SchedAuto(0);
285     return (0);
286 }
287
288 int SchdPriorFn(void)
289 {
290     HWND InfoFnWin;
291     unsigned short int * screenbucket = NULL;
```

```
292     int curpos;
293     screenbucket = SaveScreen(screenbucket, & curpos);
294     pcuron();
295     open_repair(1);
296     b4browse(b4quick_browse, b4quick_edit);
297     d4close();
298     pcuroff();
299     RestoreScreen(screenbucket, & curpos);
300     free(screenbucket);
301     return (0);
302 }
303
304 int ChngCompFn(void)
305 {
306     long i;
307     HWND UserFnWin11;
308
309     UserFnWin11 = vcreat(7, 40, ERR, YES);
310     vlocate(UserFnWin11, 9, 20);
311     vframe(UserFnWin11, ERR, FRDOUBLE);
312     vshadow(UserFnWin11, CURRENT, SHADOW100, BOTTOMRIGHT);
313     vmtitle(UserFnWin11, _TOP, CENTERJUST, ERR, " none ");
314     visible(UserFnWin11, YES, YES);
315     for (i = 0; i < 3200000; i++);
316
317
318     vdelete(UserFnWin11, NONE);
319     return (0);
320 }
321
322 int ChngCancFn(void)
323 {
324     long i;
325     HWND UserFnWin12;
326
327     UserFnWin12 = vcreat(7, 40, ERR, YES);
328     vlocate(UserFnWin12, 9, 20);
329     vframe(UserFnWin12, ERR, FRDOUBLE);
330     vshadow(UserFnWin12, CURRENT, SHADOW100, BOTTOMRIGHT);
331     vmtitle(UserFnWin12, _TOP, CENTERJUST, ERR, " none ");
332     visible(UserFnWin12, YES, YES);
333     for (i = 0; i < 3200000; i++);
334
335
336     vdelete(UserFnWin12, NONE);
337     return (0);
338 }
339
340 int ChngRestFn(void)
341 {
342     long i;
343     HWND UserFnWin13;
344
345     UserFnWin13 = vcreat(7, 40, ERR, YES);
346     vlocate(UserFnWin13, 9, 20);
347     vframe(UserFnWin13, ERR, FRDOUBLE);
348     vshadow(UserFnWin13, CURRENT, SHADOW100, BOTTOMRIGHT);
349     vmtitle(UserFnWin13, _TOP, CENTERJUST, ERR, " none ");
350     visible(UserFnWin13, YES, YES);
351     for (i = 0; i < 3200000; i++);
352
353
354     vdelete(UserFnWin13, NONE);
355     return (0);
356 }
357
358 int ChngDelAFn(void)
359 {
360     long i;
361     HWND UserFnWin14;
362
363     UserFnWin14 = vcreat(7, 40, ERR, YES);
364     vlocate(UserFnWin14, 9, 20);
365     vframe(UserFnWin14, ERR, FRDOUBLE);
```

```
366     vshadow(UserFnWin14, CURRENT, SHADOW100, BOTTOMRIGHT);
367     vmtitle(UserFnWin14, _TOP, CENTERJUST, ERR, " none ");
368     visible(UserFnWin14, YES, YES);
369     for (i = 0; i < 3200000; i++);
370
371
372     vdelete(UserFnWin14, NONE);
373     return (0);
374 }
375
376
377 int EdFlsRepFn(void)
378 {
379     unsigned short int * screenbucket = NULL;
380     int curpos;
381     screenbucket = SaveScreen(screenbucket, & curpos);
382     pcuron();
383     open_repair(1);
384     b4browse(b4quick_browse, b4quick_edit);
385     d4close();
386     pcuroff();
387     RestoreScreen(screenbucket, & curpos);
388     free(screenbucket);
389     return (0);
390 }
391
392 int EdFlsEqSupFn(void)
393 {
394     unsigned short int * screenbucket = NULL;
395     int curpos;
396     screenbucket = SaveScreen(screenbucket, & curpos);
397     pcuron();
398     open_eqpsup(1);
399     b4browse(b4quick_browse, b4quick_edit);
400     d4close();
401     pcuroff();
402     RestoreScreen(screenbucket, & curpos);
403     free(screenbucket);
404     return (0);
405 }
406 int EdFlsEqReqFn(void)
407 {
408     unsigned short int * screenbucket = NULL;
409     int curpos;
410     screenbucket = SaveScreen(screenbucket, & curpos);
411     pcuron();
412     open_eqpreq(1);
413     b4browse(b4quick_browse, b4quick_edit);
414     d4close();
415     pcuroff();
416     RestoreScreen(screenbucket, & curpos);
417     free(screenbucket);
418     return (0);
419 }
420 int EdFlsMaSupFn(void)
421 {
422     unsigned short int * screenbucket = NULL;
423     int curpos;
424     screenbucket = SaveScreen(screenbucket, & curpos);
425     pcuron();
426     open_matsup(1);
427     b4browse(b4quick_browse, b4quick_edit);
428     d4close();
429     pcuroff();
430     RestoreScreen(screenbucket, & curpos);
431     free(screenbucket);
432     return (0);
433 }
434
435 int EdFlsMaReqFn(void)
436 {
437     unsigned short int * screenbucket = NULL;
438     int curpos;
439     screenbucket = SaveScreen(screenbucket, & curpos);
```

```
440     pcuron();
441     open_matreq(1);
442     b4browse(b4quick_browse, b4quick_edit);
443     d4close();
444     pcuroff();
445     RestoreScreen(screenbucket, & curpos);
446     free(screenbucket);
447     return (0);
448 }
449
450 int EdFlsRepInfoFn(void)
451 {
452     unsigned short int * screenbucket = NULL;
453     int curpos;
454     screenbucket = SaveScreen(screenbucket, & curpos);
455     pcuron();
456     open_repinfo(1);
457     b4browse(b4quick_browse, b4quick_edit);
458     d4close();
459     pcuroff();
460     RestoreScreen(screenbucket, & curpos);
461     free(screenbucket);
462     return (0);
463 }
464
465 int EdFlsFacPrtyFn(void)
466 {
467     unsigned short int * screenbucket = NULL;
468     int curpos;
469     screenbucket = SaveScreen(screenbucket, & curpos);
470     pcuron();
471     open_facprt(1);
472     b4browse(b4quick_browse, b4quick_edit);
473     d4close();
474     pcuroff();
475     RestoreScreen(screenbucket, & curpos);
476     free(screenbucket);
477     return (0);
478 }
479
480 int EdFlsResetFn(void)
481 {
482     HWND win;
483
484     /* WINDOW STUFF */
485     win = vcreat(7, 40, EMPHNORML, YES);
486     vautoshd(win, CURRENT, SHADOW75, BOTTOMRIGHT);
487     vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Data Reset ");
488     vlocate(win, 9, 20);
489     vframe(win, EMPHNORML, FRDOUBLE);
490     visible(win, YES, YES);
491
492     vatputs(win, 1, 2, "Reseting Data Files ...");
493     vatputs(win, 3, 2, "Percent Complete :   0%");
494     d4close_all();
495     fcloseall();
496
497     open_repair(0);
498     d4zap(1L, d4r ccount());
499     d4close_all();
500     vatputs(win, 3, 2, "Percent Complete :  20%");
501
502     open_matreq(0);
503     d4zap(1L, d4reccount());
504     d4close_all();
505     vatputs(win, 3, 2, "Percent Complete :  40%");
506
507     open_eqpreq(0);
508     d4zap(1L, d4reccount());
509     d4close_all();
510     vatputs(win, 3, 2, "Percent Complete :  60%");
511
512     open_repinfo(0);
513     d4zap(1L, d4reccount());
```

```
514     d4close_all();
515     vatputs(win, 3, 2, "Percent Complete :  80%");
516
517     open_matsup(0);
518     d4top();
519     while (! d4eof())
519     {
520         f4r_double(f4ref("QTY"), 1000.0);
521         d4skip(1);
522     }
523     d4pack();
524     d4close_all();
525
526     open_eqpsup(0);
527     d4pack();
528     d4close_all();
529     fcloseall();
530
531     vatputs(win, 3, 2, "Percent Complete : 100%");
532     vatputs(win, 5, 2, "press any key ...");
533     getch();
534     vdelete(win, NONE);
535     return (0);
536 }
537
538 int GnRpSchdFn(void)
539 {
540     ReprtSched();
541     return (0);
542 }
543
544 int GnRpGantFn(void)
545 {
546     ReprtGantt();
547     return (0);
548 }
549
550 int GnRpCompFn(void)
551 {
552     long i;
553     HWND UserFnWin17;
554
555     UserFnWin17 = vcreat(7, 40, ERR, YES);
556     vlocate(UserFnWin17, 9, 20);
557     vframe(UserFnWin17, ERR, FRDOUBLE);
558     vshadow(UserFnWin17, CURRENT, SHADOW100, BOTTOMRIGHT);
559     vmtitle(UserFnWin17, _TOP, CENTERJUST, ERR, " none ");
560     visible(UserFnWin17, YES, YES);
561     for (i = 0; i < 3200000; i++);
562
563
564     vdelete(UserFnWin17, NONE);
565     return (0);
566 }
567
568 int GnRpCancFn(void)
569 {
570     long i;
571     HWND UserFnWin18;
572
573     UserFnWin18 = vcreat(7, 40, ERR, YES);
574     vlocate(UserFnWin18, 9, 20);
575     vframe(UserFnWin18, ERR, FRDOUBLE);
576     vshadow(UserFnWin18, CURRENT, SHADOW100, BOTTOMRIGHT);
577     vmtitle(UserFnWin18, _TOP, CENTERJUST, ERR, " none ");
578     visible(UserFnWin18, YES, YES);
579     for (i = 0; i < 3200000; i++);
580
581
582     vdelete(UserFnWin18, NONE);
583     return (0);
584 }
585
586 int GnRpSuspFn(void)
```

```
587 {
588     long i;
589     HWND UserFnWin19;
590
591     UserFnWin19 = vcreat(7, 40, ERR, YES);
592     vlocate(UserFnWin19, 9, 20);
593     vframe(UserFnWin19, ERR, FRDOUBLE);
594     vshadow(UserFnWin19, CURRENT, SHADOW100, BOTTOMRIGHT);
595     vmtitle(UserFnWin19, _TOP, CENTERJUST, ERR, " none ");
596     visible(UserFnWin19, YES, YES);
597     for (i = 0; i < 3200000; i++);
598
599
600     vdelete(UserFnWin19, NONE);
601     return (0);
602 }
603
604 int GnRpPossFn(void)
605 {
606     long i;
607     HWND UserFnWin20;
608
609     UserFnWin20 = vcreat(7, 40, ERR, YES);
610     vlocate(UserFnWin20, 9, 20);
611     vframe(UserFnWin20, ERR, FRDOUBLE);
612     vshadow(UserFnWin20, CURRENT, SHADOW100, BOTTOMRIGHT);
613     vmtitle(UserFnWin20, _TOP, CENTERJUST, ERR, " none ");
614     visible(UserFnWin20, YES, YES);
615     for (i = 0; i < 3200000; i++);
616
617
618     vdelete(UserFnWin20, NONE);
619     return (0);
620 }
621
622 int GnRpAMatFn(void)
623 {
624     long i;
625     HWND UserFnWin21;
626
627     UserFnWin21 = vcreat(7, 40, ERR, YES);
628     vlocate(UserFnWin21, 9, 20);
629     vframe(UserFnWin21, ERR, FRDOUBLE);
630     vshadow(UserFnWin21, CURRENT, SHADOW100, BOTTOMRIGHT);
631     vmtitle(UserFnWin21, _TOP, CENTERJUST, ERR, " none ");
632     visible(UserFnWin21, YES, YES);
633     for (i = 0; i < 3200000; i++);
634
635
636     vdelete(UserFnWin21, NONE);
637     return (0);
638 }
639
640 int GnRpUMatFn(void)
641 {
642     long i;
643     HWND UserFnWin22;
644
645     UserFnWin22 = vcreat(7, 40, ERR, YES);
646     vlocate(UserFnWin22, 9, 20);
647     vframe(UserFnWin22, ERR, FRDOUBLE);
648     vshadow(UserFnWin22, CURRENT, SHADOW100, BOTTOMRIGHT);
649     vmtitle(UserFnWin22, _TOP, CENTERJUST, ERR, " none ");
650     visible(UserFnWin22, YES, YES);
651     for (i = 0; i < 3200000; i++);
652
653
654     vdelete(UserFnWin22, NONE);
655     return (0);
656 }
657
658 int GnRpABMtFn(void)
659 {
660     long i;
```

```
661     HWND UserFnWin23;
662
663     UserFnWin23 = vcreat(7, 40, ERR, YES);
664     vlocate(UserFnWin23, 9, 20);
665     vframe(UserFnWin23, ERR, FRDOUBLE);
666     vshadow(UserFnWin23, CURRENT, SHADOW100, BOTTOMRIGHT);
667     vmtitle(UserFnWin23, _TOP, CENTERJUST, ERR, " none ");
668     visible(UserFnWin23, YES, YES);
669     for (i = 0; i < 3200000; i++);
670
671
672     vdelete(UserFnWin23, NONE);
673     return (0);
674 }
675
676 int GnRpABEqFn(void)
677 {
678     long i;
679     HWND UserFnWin24;
680
681     UserFnWin24 = vcreat(7, 40, ERR, YES);
682     vlocate(UserFnWin24, 9, 20);
683     vframe(UserFnWin24, ERR, FRDOUBLE);
684     vshadow(UserFnWin24, CURRENT, SHADOW100, BOTTOMRIGHT);
685     vmtitle(UserFnWin24, _TOP, CENTERJUST, ERR, " none ");
686     visible(UserFnWin24, YES, YES);
687     for (i = 0; i < 3200000; i++);
688
689
690     vdelete(UserFnWin24, NONE);
691     return (0);
692 }
```

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include <stdlib.h>
 4
 5 #include "g4char.h"
 6 #include "w4.h"
 7 #include "d4base.h"
 8 #include "PWRBASE.H"
 9 #include "LKUmod.H"
10 #include "STR.H"
11
12 #define NO_ERROR
13
14 /* Local Prototypes ************************************************************/
15
16 long  LKUmodNextField(long, int);
17 void  LKUmodGetRefs(void);
18 int   LKUmodLookWindowWidth(void);
19 int   LKUmodfield_width(long);
20 int   LKUmodall_fields(int);
21 void  LKUmodDisplayRow(int, long);
22 void  LKUmodDisplayLookup(int);
23 void  LKUmodPositionLookup(long);
24
25 /* Globals *********************************************************************/
26
27 extern copyright;
28 extern COPYRIGHT_NOTICE();
29
30 static LOOKUP LKUmodLookup;
31
32 static int    LKUmodfirstTime = 1;
33 static int    LKUmodshowFields;
34 static int    LKUmodcurrentRef;
35 static int    LKUmodtotlWndwWidth;
36 static int    LKUmodtotalFields;
37 static int    LKUmodfrceCols;
38 static int    LKUmodallowExtra = 1;
39 static char   LKUmoddateFormat[30];
40 static int    LKUmodscrollFields;
41 static int    LKUmodtruncOutput;
42
43 static int    LKUmodwindowWidth;
44 static long   LKUmodfieldRefs[TOTALFIELDS];
45 static int    LKUmodwidths[TOTALFIELDS];
46
47
48 long LKUmodNextField(long fRef, int curIndex)
49 {
50     int  found;
51     int  i = 0;
52
53     if (! LKUmodshowFields)
54         return (long) fRef;
55     found = 0;
56     while (i < curIndex)
56     {
57         if (LKUmodfieldRefs[i] == fRef)
57         {
58             found = 1;
59             break;
60         }
61         i++;
62     }
63     if (! found)
64         return (long) fRef;
65     return (long) - 1;
66 }
67
68 int LKUmodset_columns(int columnWidth)
69 {
70     int i = 0;
71
72     if (columnWidth != -1)
```

```
    {
        if (columnWidth > CRTWIDTH || columnWidth < 0)
            columnWidth = 0;

        while (i < TOTALFIELDS)
        {
            LKUmodwidths[i] = columnWidth;
            i++;
        }
    }
    return (int) LKUmodwidths[i];
}

void LKUmodreset_columns(void)
{
    LKUmodset_columns(0);
}

void LKUmodGetRefs(void)
{
    int  i = LKUmodshowFields;
    int  j = 0;
    int  startNum;

    LKUmodtotalFields = f4num_fields();

    if (LKUmodshowFields)
        while (j < i)
        {
            LKUmodfieldRefs[j] = f4ref(LKUmodLookup.lField[j].fieldName);
            if (! LKUmodwidths[j] || LKUmodwidths[j] == -2)
                if (f4type(LKUmodfieldRefs[j]) == 'M')
                    LKUmodwidths[j] = 4;
                else
                    if (f4type(LKUmodfieldRefs[j]) == 'D')
                        LKUmodwidths[j] = strlen(LKUmoddateFormat);
                    else
                        if (LKUmodwidths[j] == -2)
                            LKUmodwidths[j] = - (f4width(LKUmodfieldRefs[j]));
                        else
                            LKUmodwidths[j] = f4width(LKUmodfieldRefs[j]);

            j++;
        }
    if (LKUmodtotalFields >= TOTALFIELDS)
        LKUmodtotalFields = TOTALFIELDS - 1;
    while (i < LKUmodtotalFields)
    {
        LKUmodfieldRefs[i] = LKUmodNextField(f4j_ref(i + 1), i);
        if (LKUmodfieldRefs[i] == -1)
        {
            startNum = 0;
            while (startNum < LKUmodtotalFields)
            {
                if ((LKUmodfieldRefs[i] = LKUmodNextField((long) startNum, (int) LKUmodtotalFie
lds)) != (long) - 1)
                    break;
                startNum++;
            }
        }
        if (! LKUmodwidths[i])
            if (f4type(LKUmodfieldRefs[i]) == 'M')
                LKUmodwidths[i] = 4;
            else
                if (f4type(LKUmodfieldRefs[i]) == 'D')
                    LKUmodwidths i] = strlen(LKUmoddateFormat);
                else
                    LKUmodwidths[i] = f4width(LKUmodfieldRefs[i]);
        if (! LKUmodshowFields && i < MAXLKUFIELDS)
            LKUmodLookup.lField[i].attribute = LKUmodLookup.windowAttr;
        i++;
    }
}
```

```
140 int  LKUmodLookWindowWidth(void)
141 {
142     int  i;
143     int  totalWidth = 1, tWidth1, tWidth2, lastWidth, joinField;
144
145     for (i = 0; i < ((LKUmodshowFields > 0) ? LKUmodshowFields : MAXLKUFIELDS); i++)
145     {
146
147         tWidth1 = LKUmodfield_width(LKUmodfieldRefs[i]) + 1;
148         if (LKUmodwidths[i] < 0)
148         {
149             tWidth2 = - (LKUmodwidths[i]) + 1;
150             joinField = 1;
151             while ((LKUmodwidths[i] < 0) &&
152                     (i < ((LKUmodshowFields > 0) ? LKUmodshowFields : MAXLKUFIELDS)))
153                     i++;
154         }
154         else
154         {
155             tWidth2 = LKUmodwidths[i] + 1;
156             joinField = 0;
157         }
158
159         if (! LKUmodfrceCols)
159         {
160             if (tWidth2 && joinField)
160             {
161                 lastWidth = tWidth2;
162             }
162             else
162             {
163                 if (tWidth1 < tWidth2)
163                 {
164                     lastWidth = tWidth1;
165                 }
165                 else
165                 {
166                     lastWidth = tWidth2;
167                 }
168             }
169         }
169         else
170             lastWidth = tWidth2;
171
172         totalWidth += lastWidth;
173
174         if (totalWidth > CRTWIDTH)
174         {
175             totalWidth -= ((totalWidth - CRTWIDTH) + 1);
176             break;
177         }
178     }
179
180     return (totalWidth + 1);
181 }
182
183 int  LKUmodfield_width(long fieldRef)
184 {
185     int length;
186
187     if (f4type(fieldRef) == 'M')
188         length = 4;
189     else
190         if (f4type(fieldRef) == 'D')
191             length = strlen(LKUmoddateFormat);
192         else
193             length = f4width(fieldRef);
194
195     return length;
196 }
197
198 int  LKUmodall_fields(int refOffset)
199 {
200     int i = 0, tFields = 0, currntPos = 0, reachedEnd = 0;
```

```
201     int tWidth1, tWidth2, joinField;
202
203     while (1)
203     {
204         if ((refOffset + i) >= LKUmodtotalFields)
204         {
205             reachedEnd = 1;
206             break;
207         }
208         if (currntPos >= LKUmodtotlWndwWidth)
209             break;
210         tWidth1 = LKUmodfield_width(LKUmodfieldRefs[refOffset + i]) + 1;
211         if (LKUmodwidths[refOffset + i] < 0)
211         {
212             tWidth2 = - (LKUmodwidths[refOffset + i]) + 1;
213             joinField = 1;
214             while ((LKUmodwidths[refOffset + i] < 0) &&
215                     (i < ((LKUmodshowFields > 0) ? LKUmodshowFields : MAXLKUFIELDS)))
216                     i++;
217         }
217         else
217         {
218             tWidth2 = LKUmodwidths[refOffset + i] + 1;
219             joinField = 0;
220         }
221
222         if (! LKUmodfrceCols)
222         {
223             if (tWidth2 && joinField)
223             {
224                 currntPos += tWidth2;
225             }
225             else
225             {
226                 if (tWidth1 < tWidth2)
226                 {
227                     currntPos += tWidth1;
228                 }
228                 else
228                 {
229                     currntPos += tWidth2;
230                 }
231             }
232         }
232         else
233             currntPos += tWidth2;
234
235         tFields++;
236         i++;
237     }
238     return reachedEnd;
239 }
240
241 void LKUmodDisplayRow(int row, long highLight)
242 {
243     int   i = 0;
244     int   colPos = 0, prevPos;
245     int   nextWidth;
246 /*   int   tWidth ; */
247     long  fieldRef;
248 /*   int   frceWidth ;*/
249     int   padLength;
250     int   joinField;
251     int   nextOffset;
252     int   attrib, trimLen;
253     int   firstCombo;
254     char  outputText[CRTWIDTH];
255     char  holdText[CRTWIDTH];
256     char  joinString[CRTWIDTH];
257     char  join2String[CRTWIDTH];
258     char  fieldText[MAXLKUFIELDWIDTH];
259     firstCombo = 0;
260     nextOffset = 0;
261     joinField = 0;
```

```
262     strcpy(joinString, "\0");
263     strcpy(join2String, "\0");
264     while (1)
264     {
265         prevPos = colPos;
266
267         if ((LKUmodcurrentRef + i) >= LKUmodtotalFields)
268             break;
269
270         if (! nextOffset && LKUmodcurrentRef > 0)
271             if (LKUmodwidths[LKUmodcurrentRef + i - 1] < 0)
271             {
272                 i++;
273                 continue;
274             }
275
276         fieldRef = LKUmodfieldRefs[LKUmodcurrentRef + i];
277         nextWidth = LKUmodwidths[LKUmodcurrentRef + i];
278
279         /* check for field combinations */
280         if (nextWidth < 0)
280         {
281             joinField = 1;
282             nextWidth *= -1;
283             strcpy(join2String, "More");
284             firstCombo++;
285         }
285         else
285         {
286             strcpy(join2String, "\0");
287         }
288
289         if (colPos >= (LKUmodtotlWndwWidth - 1))
290             break;
291
292         if (! nextOffset)
293 /*          w4num_att( row, colPos++, " ", 1, (highLight == 0) ?
294             LKUmodLookup.lField[ (LKUmodshowFields && i >= LKUmodshowFields)
295                                 ? LKUmodshowFields-1 : i ].attribute
296             : highLight ) ;
297 */
298             w4num_att(row, colPos++, " ", 1, (highLight == 0) ?
299                     LKUmodLookup.lField[i ].attribute
300                     : highLight);
301
302
303         if (LKUmodshowFields > 0 && i >= LKUmodshowFields)
304             if (! LKUmodallowExtra)
305                 break;
306         if ((i + 1) > ((LKUmodtotalFields < MAXLKUFIELDS) ? LKUmodtotalFields : MAXLKUFIELDS))
307                 break;
308
309         if (LKUmodallowExtra && i >= LKUmodshowFields)
310             LKUmodLookup.lField[i].attribute = LKUmodLookup.windowAttr;
311
312         if (f4type(fieldRef) == 'D')
313             if (f4str(fieldRef)[0] == ' ')
314                 strcpy(fieldText, STRspaces(strlen(LKUmoddateFormat)));
315             else
316                 strcpy(fieldText, c4dt_format(f4str(fieldRef), LKUmoddateFormat));
317         else
318             if (f4type(fieldRef) == 'M')
319                 strcpy(fieldText, "MEMO");
320             else
321                 strcpy(fieldText, f4str(fieldRef));
322
323
324         if (joinField)
324         {
325             if (join2String[0] != '\0')
325             {
326                 strcpy(join2String, STRrtrim(fieldText));
327                 strcat(join2String, LKUmodLookup.lField[LKUmodcurrentRef + i].joinChars);
328                 strcat(joinString, join2String);
```

```
329                 if (strlen(joinString) > CRTWIDTH)
330                     strcpy(holdText, STRleft(joinString, CRTWIDTH - 1));
331                 else
332                     strcpy(holdText, joinString);
333                 nextOffset = strlen(holdText);
334                 i++;
335                 continue;
336             }
337             strcpy(joinString, "\0");
338
339             if ((strlen(fieldText) + strlen(holdText)) < CRTWIDTH)
340                 strcat(holdText, STRrtrim(fieldText));
341             else
342                 strcat(holdText,
343                     STRleft(STRrtrim(fieldText), ((CRTWIDTH - 1) - strlen(holdText))));
344
345             strcpy(fieldText, holdText);
346
347             nextWidth = - (LKUmodwidths[i - firstCombo]);
348             if (nextWidth < 0) nextWidth *= -1;
349
350             firstCombo = 0;
351             joinField = 0;
352             nextOffset = 0;
353         }
354
355         strcpy(outputText, "\0");
356         if (nextWidth)
356         {
357             padLength = nextWidth - strlen(fieldText);
358             if ((padLength > 0) && LKUmodfrceCols)
359                 if (padLength < CRTWIDTH)
360                     strcpy(outputText, STRspaces(padLength));
361                 else
362                     strcpy(outputText, STRleft(STRspaces(padLength), CRTWIDTH - 1));
363             else
364                 if ((int) nextWidth > (int) strlen(fieldText))
365                     nextWidth = strlen(fieldText);
366         }
366         else
367             nextWidth = strlen(fieldText);
368         if ((strlen(outputText) + strlen(fieldText)) < CRTWIDTH)
369             strcat(outputText, fieldText);
370         else
371             strcat(outputText, STRleft(fieldText, CRTWIDTH - 1));
372
373         if ((int) strlen(outputText) < (int) nextWidth)
374             strcat(outputText, STRspaces(nextWidth - strlen(outputText)));
375         else
376             outputText[nextWidth] = '\0';
377
378         trimLen = strlen(outputText) - (LKUmodtotlWndwWidth - colPos);
379         if (trimLen > 0)
380             trimLen = strlen(outputText) - trimLen;
381
382         if (highLight == 0)
383             attrib = (int) LKUmodLookup.lField[i].attribute;
384         else
385             attrib = (int) highLight;
386         if (trimLen > 0)
387             if (LKUmodtruncOutput)
388                 w4num_att(row, colPos, STRleft(outputText, trimLen),
389                     trimLen, (long) attrib);
390             else
390             {
391                 w4num_att(row, colPos, STRleft(STRspaces(trimLen + i), trimLen),
392                     trimLen, (long) attrib);
393             }
394         else
395             w4num_att(row, colPos, outputText, nextWidth, (long) attrib);
396
397         colPos += nextWidth;
398         i++;
399     }
```

```
400
401     if (prevPos < LKUmodtotlWndwWidth)
402         w4num_att(row, prevPos, STRspaces((LKUmodtotlWndwWidth - prevPos) + 1),
403             (LKUmodtotlWndwWidth - prevPos),
404             (highLight == 0) ? LKUmodLookup.windowAttr : highLight);
405
406 }
407
408 void LKUmodDisplayLookup(int row)
409 {
410     long cr = d4recno ();
411     int i;
412
413     x4skip((long) - row);
414     for (i = 0; i < LKUmodLookup.height; i++, x4skip(1L))
414     {
415         LKUmodDisplayRow(i, (long) 0);
416     }
417     d4go(cr);
418 }
419
420 void LKUmodPositionLookup(long records)
421 {
422     x4skip(records);
423 }
424
425 long  LKUmodlookup(void)
426 {
427
428     long d4bof_BUG_FIX;
429
430     int   lookWindow;
431     int   lookWidth;
432     int   row;
433     int   returnKey = 0;
434     long  SelectRecord;
435     char  tempString1[81];
436
437     /* Ensure that the LKUmodinit function was called first */
438     if (LKUmodfirstTime)
438     {
439         u4error(1200, "Uninitialized Structure", "",
440             "LKUmodinit() must be called prior to LKUmodlookup()", (char *) 0);
441         return 0;
442     }
443
444     /* Open / Select the file(s) to lookup */
445     if (LKUmodLookup.lfile.dbfSelect >= 0)
445     {
446
447         if (d4select(LKUmodLookup.lfile.dbfSelect) < 0)
448             return 0;
449         if (LKUmodLookup.lfile.ndxSelect >= 0)
450             if (i4select(LKUmodLookup.lfile.ndxSelect) < 0)
451                 return 0;
452     }
452     else
452     {
453         if (d4use(LKUmodLookup.lfile.fileName) < 0)
453         {
454          #ifndef NO_ERROR
455          /* Problem occured when closing the answer database */
456             sprintf(tempString1,"Error when opening %s",
457                LKUmodLookup.lfile.fileName) ;
458             u4error(857,tempString1, "", (char *)0 ) ;
459          #endif
460             return 0;
461         }
462         if (LKUmodLookup.lfile.indxName[0] != '\0')
463             if (i4open(LKUmodLookup.lfile.indxName) < 0)
464                 return 0;;
465     }
466
467
```

```
468
469     if (LKUmodLookup.filterRoutine != NULL)
470         x4filter(LKUmodLookup.filterRoutine);
471
472     x4top();
473     d4bof_BUG_FIX = d4recno();
474
475     LKUmodGetRefs();
476     lookWidth = LKUmodtotlWndwWidth = (LKUmodwindowWidth) ? LKUmodwindowWidth
477         : LKUmodLookWindowWidth();
478     lookWindow = w4define(LKUmodLookup.startRow, LKUmodLookup.startCol,
479         LKUmodLookup.startRow + LKUmodLookup.height + 1, LKUmodLookup.startCol + lookWidth
    ;
480     w4attribute(LKUmodLookup.windowAttr);
481 /*
482         if( LKUmodLookup.shadow )
483       w4shadow( lo kWindow, ON ) ;
484 */
485     w4popup();
486     w4border(DOUBLE_TOP, LKUmodLookup.windowAttr);
487
488     /* Check the title strings' length... if greater than the window
489        then cut it */
490     strcpy(tempString1, LKUmodLookup.title);
491     if ((int) strlen(tempString1) > (int) lookWidth)
492         tempString1[lookWidth] = '\0';
493     w4title((int) - 1, (int) - 1, (char *) tempString1, (long) LKUmodLookup.windowAttr);
494
495     w4activate(lookWindow);
496     w4clear(0);
497     w4cursor(-1, -1);
498
499
500     row = 0;
501     LKUmodDisplayLookup(row);
502
503     while ((returnKey != RETURN) && (returnKey != ESC) && (returnKey != F3))
503     {
504         LKUmodDisplayRow(row, LKUmodLookup.highLightBar);
505         returnKey = g4char();
506         LKUmodDisplayRow(row, (long) 0);
507         switch (returnKey)
507         {
508         case ESC : SelectRecord = -1L;
509             break;
510         case F3 : SelectRecord = -2L;
511             break;
512         case RETURN : SelectRecord = d4recno();
513             break;
514
515         case F2 :                       /* get matid number to seek on                    */
516             /* maybe incorporate string seek on matdesc */
517             break;
518
519         case RIGHT :
520             if (LKUmodscrollFields)
521                 if (! LKUmodall_fields(LKUmodcurrentRef))
521                 {
522                     while (1)
522                     {
523                         LKUmodcurrentRef++;
524                         if (LKUmodwidths[LKUmodcurrentRef - 1] >= 0 ||
525                                 LKUmodcurrentRef >= LKUmodtotalFields)
526                             break;
527                     }
528                     LKUmodDisplayLookup(row);
529                 }
530             break;
531
532         case LEFT :
533             if (LKUmodscrollFields)
534                 if (LKUmodcurrentRef)
534                 {
535                     while (1)
```

```
535                     {
536                         LKUmodcurrentRef--;
537                         if (LKUmodwidths[LKUmodcurrentRef - 1] >= 0 ||
538                                 LKUmodcurrentRef == 0)
539                             break;
540                     }
541                     LKUmodDisplayLookup(row);
542                 }
543             break;
544
545         case UP : if (d4recno() != d4bof_BUG_FIX)
545             {
546                 LKUmodPositionLookup(-1L);
547                 if (row > 0)
548                     row--;
549                 else
550                     if (! d4bof())
551                         w4scroll(-1);
552             }
553             break;
554
555         case DOWN : LKUmodPositionLookup(1L);
556             if (++row == LKUmodLookup.height || d4eof())
556             {
557                 if (! d4eof())
557                 {
558                     w4scroll(1);
559                 }
559                 else
560                     x4bottom();
561                 row--;
562             }
563             break;
564
565         case PGDN : LKUmodPositionLookup((long) (LKUmodLookup.height - 1));
566             if (d4eof())
566             {
567                 LKUmodPositionLookup((long) (- LKUmodLookup.height));
568                 if (d4bof())
569                     x4top();
570             }
571             row = 0;
572             LKUmodDisplayLookup(row);
573             row = 0;
574             break;
575
576         case PGUP : LKUmodPositionLookup((long) (- LKUmodLookup.height));
577             if (d4bof())
578                 x4top();
579             row = 0;
580             LKUmodDisplayLookup(row);
581             row = 0;
582             break;
583
584         case HOME : x4top();
585             row = 0;
586             LKUmodDisplayLookup(row);
587             row = 0;
588             break;
589
590         case END : x4bottom();
591             x4skip((long) - (LKUmodLookup.height - 1));
592             row = 0;
593             LKUmodDisplayLookup(row);
594             row = 0;
595             break;
596
597         }
598     }
599
600     w4deactivate(lookWindow);
601     w4close(lookWindow);
602
603     if (LKUmodLookup.lfile.dbfSelect < 0)
```

```
604         if (d4close())
604         {
605          #ifndef NO_ERROR
606          /* Problem occured when closing the query database */
607             sprintf(tempString1,"Error when closing %s",
608               LKUmodLookup.lfile.fileName ) ;
609             u4error(858,tempString1, "", (char *)0 ) ;
610          #endif
611         }
612
613     return SelectRecord;
614 }
615
616
617 void LKUmoddate_format(char * date_format)
618 {
619     if (strlen(date_format) < 30)
620         strcpy(LKUmoddateFormat, date_format);
621 }
622
623 void LKUmodfield(char * fName, long attr)
624 {
625     int  fieldLength = strlen (fName);
626     char tempString1[81];
627     char tempString2[81];
628
629     if (fieldLength > 21 && fName != (char *) 0)
629     {
630       #ifndef NO_ERROR
631          u4error(1210,"Field name too long", "", fName, "",
632             "Must be 10 characters of less", "", tempString1, (char *)0 ) ;
633       #endif
634     }
634     else
634     {
635         strcpy(LKUmodLookup.lField[LKUmodshowFields].fieldName, fName);
636         if (attr >= 0L && attr <= 255L)
637             LKUmodLookup.lField[LKUmodshowFields].attribute = attr;
638
639       #ifndef NO_ERROR
640       else {
641          sprintf(tempString1,"%d",(int)attr) ;
642          sprintf(tempString2,"Using the default field attribute value of %d",
643             (int)DEFAULT_FATTR) ;
644          u4error(1220,"Invalid attribute value", "", tempString1, "",
645             "Attribute must be between 0 and 255", "", tempString2, (char *)0 ) ;
646       }
647       #endif
648
649         if (fName != (char *) 0)
650             LKUmodshowFields++;
651     }
652 }
653
654 void LKUmodinit(void)
655 {
656     strcpy(LKUmodLookup.title, DEFAULT_TITLE);
657     LKUmodLookup.windowAttr = (long) DEFAULT_WATTR;
658     LKUmodLookup.startRow = (char) STARTROW;
659     LKUmodLookup.startCol = (char) STARTCOL;
660     LKUmodLookup.height = (char) DEFAULT_HEIGHT;
661     LKUmodLookup.highLightBar = (long) DEFAULT_HATTR;
662     strcpy(LKUmodLookup.lfile.fileName, "\0");
663     strcpy(LKUmodLookup.lfile.indxName, "\0");
664     LKUmodLookup.lfile.dbfSelect = -1;
665     LKUmodLookup.lfile.ndxSelect = -1;
666     LKUmodLookup.filterRoutine = NULL;
667     LKUmoddate_format("MM/DD/YY");
668     LKUmodfrceCols = 0;
669     LKUmodwindowWidth = 0;
670     LKUmodallowExtra = 1;
671     LKUmodLookup.shadow = 0;
672     LKUmodset_columns(0);
673     LKUmodscrollFields = 1;
```

```
674     LKUmodtruncOutput = 1;
675
676     LKUmodcurrentRef = LKUmodshowFields = LKUmodfirstTime = 0;
677 /*
678         if( ! copyright )
679                 COPYRIGHT_NOTICE() ;
680 */
681 )
682
683 void LKUmodtitle(char * title)
684 {
685     int  titleLength = strlen (title);
686     char tempString1[81];
687     char tempString2[81];
688
689     if (titleLength > TITLELENGTH)
689     {
690       #ifndef NO_ERROR
691          sprintf(tempString1,"Must be %d characters of less",TITLELENGTH) ;
692          sprintf(tempString2,"Lookup title will contain %s",DEFAULT_TITLE) ;
693          u4error(1230,"Title too long", "", title, "", tempString1, "",
694             tempString2, (char *)0 ) ;
695       #endif
696     }
696     else
697         strcpy(LKUmodLookup.title, title);
698 )
699
700 void LKUmodposition(int startRow, int startCol, int windowHeight)
701 {
702     char tempString0[81];
703     char tempString1[81];
704     char tempString2[81];
705
706     if (startRow != -1)
706     {
707         if (startRow >= 0 && startRow <= MAXROWS)
708             LKUmodLookup.startRow = (char) startRow;
709         else
709         {
710          #ifndef NO_ERROR
711             sprintf(tempString1,"Must be between 0 and %d",MAXROWS) ;
712             sprintf(tempString2,"Using default value of %d rows",
713                (int)DEFAULT_ROWS) ;
714             u4error(1240,"Row position outside screen limits", "",
715               tempString1, "", tempString2, (char *)0 ) ;
716          #endif
717         }
718     }
719     if (startCol != -1)
719     {
720         if (startCol >= 0 && startCol <= MAXCOLS)
721             LKUmodLookup.startCol = (char) startCol;
722         else
722         {
723          #ifndef NO_ERROR
724             sprintf(tempString1,"Must be between 0 and %d",MAXCOLS) ;
725             sprintf(tempString2,"Using default value of %d columns",
726                (int)DEFAULT_COLS) ;
727             u4error(1242,"Column position outside screen limits", "",
728                tempString1, "", tempString2, (char *)0 ) ;
729          #endif
730         }
731     }
732     if (windowHeight != -1)
732     {
733         if (windowHeight >= 1 && windowHeight <= (MAXROWS - startRow))
734             LKUmodLookup.height = (char) windowHeight;
735         else
735         {
736          #ifndef NO_ERROR
737             sprintf(tempString0,"Height of %d will not fit in the window",
738                windowHeight) ;
739             sprintf(tempString1,"Must be between 1 and %d",
```

```
740                 (int)(MAXROWS-startRow)) ;
741              sprintf(tempString2,"Using default value of %d rows",
742                 (int)DEFAULT_HEIGHT) ;
743              u4error(1244,tempString0, "", tempString1, "", tempString2,
744                 (char *)0 ) ;
745           #endif
746           }
747     }
748 }
749
750 void LKUmodselect(int dbfRef, int ndxRef)
751 {
752     char tempString1[81];
753
754     if (dbfRef != -1)
754     {
755         if (d4select(dbfRef) < 0)
755         {
756          #ifndef NO_ERROR
757             sprintf(tempString1,
758                "Database base reference %d not available.",dbfRef) ;
759             u4error(1250,"Invalid Base Reference", "", tempString1,
760                "", "Exiting lookup function", (char *)0 ) ;
761          #endif
762         }
762         else
763             LKUmodLookup.lfile.dbfSelect = dbfRef;
764     }
764     else
764     {
765       #ifndef NO_ERROR
766          sprintf(tempString1,"No database base reference given.",dbfRef) ;
767          u4error(1251,"Invalid Base Reference", "",
768             "No database base reference given.", (char *)0 ) ;
769       #endif
770     }
771
772     if (ndxRef != -1)
772     {
773         if (i4select(ndxRef) < 0)
773         {
774          #ifndef NO_ERROR
775             sprintf(tempString1,"Index reference %d not available.",ndxRef) ;
776             u4error(1252,"Invalid Index Reference", "", tempString1,
777                "", "Exiting lookup function", (char *)0 ) ;
778          #endif
779         }
779         else
780             LKUmodLookup.lfile.ndxSelect = ndxRef;
781     }
781     else
782         LKUmodLookup.lfile.ndxSelect = ndxRef;
783 }
784
785 void LKUmodfile(char * dbfName, char * ndxName)
786 {
787     char fileMatch[14];
788     char tempString1[81];
789
790     if (dbfName[0] != '\0')
790     {
791         if (u4file_first(dbfName, fileMatch))
791         {
792          #ifndef NO_ERROR
793             sprintf(tempString1,"Invalid %s database path or file name.",
794                dbfName) ;
795             u4error(1254,tempString1, "", "Exiting lookup function",
796                (char *)0 ) ;
797          #endif
798         }
798         else
798         {
799             strcpy(LKUmodLookup.lfile.fileName, dbfName);
800         }
```

```
801     }
801     else
801     {
802       #ifndef NO_ERROR
803          sprintf(tempString1,"No database path or file name given.",
804                                  dbfName) ;
805                          u4error(1255,tempString1, "", "Exiting lookup function",
806                                  (char *)0 ) ;
807                  #endif
808     }
809
810     if (ndxName[0] != '\0')
810     {
811         if (u4file_first(ndxName, fileMatch))
811         {
812                         #ifndef NO_ERROR
813                                 sprintf(tempString1,"Invalid %s index path or file name.",
814                                         ndxName) ;
815                                 u4error(1256,tempString1, "", "Exiting lookup function",
816                                         (char *)0 ) ;
817                         #endif
818         }
818         else
818         {
819             strcpy(LKUmodLookup.lfile.indxName, ndxName);
820         }
821     }
822 }
823
824 void LKUmodattributes(long windowAttrib, long highlightBar)
825 {
826     char tempString1[81];
827     char tempString2[81];
828
829     if (windowAttrib != -1)
829     {
830         if (windowAttrib >= 0L && windowAttrib <= 255L)
831             LKUmodLookup.windowAttr = windowAttrib;
832
833       #ifndef NO_ERROR
834       else {
835          sprintf(tempString1,"%d",(int)windowAttrib) ;
836          sprintf(tempString2,
837             "Using the default field attribute value of %d",
838             (int)DEFAULT_FATTR) ;
839          u4error(1220,"Invalid attribute value", "", tempString1, "",
840             "Attribute must be between 0 and 255", "", tempString2,
841             (char *)0 ) ;
842       }
843       #endif
844     }
845     if (highlightBar != -1)
845     {
846         if (highlightBar >= 0L && highlightBar <= 255L)
847             LKUmodLookup.highLightBar = highlightBar;
848
849       #ifndef NO_ERROR
850       else {
851          sprintf(tempString1,"%d",(int)highlightBar) ;
852          sprintf(tempString2,"Using the default field attribute value of %d",
853             (int)DEFAULT_FATTR) ;
854          u4error(1220,"Invalid attribute value", "", tempString1, "",
855             "Attribute must be between 0 and 255", "", tempString2,
856             (char *)0 ) ;
857       }
858       #endif
859     }
860 }
861
862 void LKUmodfilter(int (* fltrFunction) (void))
863 {
864     LKUmodLookup.filterRoutine = fltrFunction;
865 }
866
```

```
867
868 int LKUmodcolumn_width(int columnNum, int width)
869 {
870     char tempString0[81];
871     char tempString1[81];
872
873     if (columnNum >= TOTALFIELDS || columnNum < 0)
873     {
874       #ifndef NO_ERROR
875          sprintf(tempString0,"Invalid column number %d",(int)columnNum) ;
876          sprintf(tempString1,"Column number must be between 0 and %d",
877             (int)TOTALFIELDS) ;
878          u4error(1260,tempString0, "", tempString1, "",
879             "Using the field width as the default", (char *)0 ) ;
880       #endif
881         return - 1;
882     }
883     if (width != -1)
883     {
884         if (width <= 0 || width > CRTWIDTH)
884         {
885          #ifndef NO_ERROR
886             sprintf(tempString0,"Invalid column width: %d",(int)width) ;
887             sprintf(tempString1,"Column width must be between 1 and %d",
888                (int)CRTWIDTH) ;
889             u4error(1262,tempString0, "", tempString1, "",
890                "Using the field width as the default", (char *)0 ) ;
891          #endif
892             return - 1;
893         }
894         if (LKUmodwidths[columnNum] < 0)
895             LKUmodwidths[columnNum] = - width;
896         else
897             LKUmodwidths[columnNum] = width;
898     }
899
900     if (LKUmodwidths[columnNum] < 0)
901         return - (LKUmodwidths[columnNum]);
902     else
903         return LKUmodwidths[columnNum];
904 }
905
906 int LKUmodforce_columns(int fcSwitch)
907 {
908     if (fcSwitch != -1)
909         LKUmodfrceCols = (fcSwitch) ? 1 : 0;
910
911     return LKUmodfrceCols;
912 }
913
914 int LKUmodallow_extra(int aeSwitch)
915 {
916     if (aeSwitch != -1)
917         LKUmodallowExtra = (aeSwitch) ? 1 : 0;
918
919     return LKUmodallowExtra;
920 }
921
922 int LKUmodwindow_width(int newWidth)
923 {
924     if (newWidth != -1)
925         if (newWidth > 0)
926             LKUmodwindowWidth = (newWidth <= CRTWIDTH) ? newWidth : 0;
927
928     return LKUmodwindowWidth;
929 }
930
931 int LKUmodscrolling(int scrollSwitch)
932 {
933     if (scrollSwitch != -1)
934         LKUmodscrollFields = (scrollSwitch) ? 1 : 0;
935
936     return LKUmodscrollFields;
937 }
```

```
938
939 int LKUmodtruncate(int truncSwitch)
940 {
941     if (truncSwitch != -1)
942         LKUmodtruncOutput = (truncSwitch) ? 1 : 0;
943
944     return LKUmodtruncOutput;
945 }
946
947 void LKUmodcombine_fields(char * joinchars)
948 {
949     if (LKUmodshowFields > 0 && LKUmodshowFields < MAXLKUFIELDS)
949     {
950         if (strlen(joinchars) > MAXJOINCHARS)
951             return;
952         if (LKUmodwidths[LKUmodshowFields - 1] == 0)
953             LKUmodwidths[LKUmodshowFields - 1] = -2;
954         else
955             LKUmodwidths[LKUmodshowFields - 1] *= -1;
956         strcpy(LKUmodLookup.lField[LKUmodshowFields - 1].joinChars, joinchars);
957     }
958 }
959
960 int LKUmodshadow(int ofSwitch)
961 {
962     if (ofSwitch != -1)
963         LKUmodLookup.shadow = (ofSwitch) ? 1 : 0;
964
965     return LKUmodLookup.shadow;
966 }
```

```
 1 /*
 2  *  main.c
 3  *
 4 */
 5
 6 #include "dwmenu.h"
 7 #include "rmrsinit.h"
 8 #include "d4base.h"
 9 #include "dw.h"
10
11 #define MAINMODULE
12 #include "rmrs.h"
13
14 void main(int argc, char * argv[])
15 {
16     if (argc > 1) setdwlib(-1, 7, -1, -1);
17     else setdwlib(-1, 3, -1, -1);
18     d4initialize(10, 15, 80, 3000, 0xFC00L);
19     IntroScreen();
20     SetupMenu();
21     while (1)
22     {
23         MNUDoSelect(mmain, NULLF);
24     }
25 }
```

```
 1 /*
 2  * miscutil.c
 3  *
 4  *     miscellaneous utility functions
 5  */
 6
 7 #include <string.h>
 8 #include <stdio.h>
 9 #include <conio.h>
10 #include <ctype.h>
11 #include "dw.h"
12
13 void trimstr(char * s)
13 {
14 /*
15  * ARGUMENT
16  *   (char *) s - input string
17  *
18  * DESCRIPTION
19  *   Trim string of leading and trailing blanks
20  *
21  * RETURNS
22  *
23  */
24
25     int len, cnt = 0, i;
26
27     /* remove white space at beginning of string */
28     for (cnt = 0; isspace(*(s + cnt)); cnt++);
29     len = strlen(s);
30     for (i = 0; i <= (len - cnt); i++)
31         * (s + i) = * (s + i + cnt);
32     * (s + len + 1) = '\0';
33
34     /* remove white space at end of string */
35     len = strlen(s);
36     len--;
37     while (isspace(*(s + len)))
38         len--;
39     * (s + len + 1) = '\0';
40
41 }
42 void dmpxtrakeys(void)
42 {
43 /*
44  * ARGUMENT
45  *
46  * DESCRIPTION
47  *   Dumps extra keystrokes from input buffer
48  *
49  * RETURNS
50  *
51  */
52
53     while (kbhit()) getch();
54 }
55
56 char * uppercase(char * str)
56 {
57 /*
58  * ARGUMENT
59  *   (char *) str - input string
60  *
61  * DESCRIPTION
62  *   Converts all lowercase characters of string to uppercase
63  *
64  * RETURNS
65  *   (char *) converted string
66  *
67  */
68
69     int i, len;
70
71     len = strlen(str);
```

```
72      for (i = 0; i <= (len - 1); i++)
73          str[i] = toupper(str[i]);
74      return (str);
75  }
```

```
 1 /*
 2  *  procauto.c
 3  *
 4  *    contains DoProcessAuto()
 5  *
 6  *      by Chris Sawyer (last rev: 1/4/91)
 7  */
 8
 9 /*
10  *      Rev 1.0 ?? Jan 1992
11  *   Original release
12  *
13  *
14  */
15
16 #include <stdio.h>
17 #include <conio.h>
18 #include <stdlib.h>
19 #include <dir.h>
20 #include <string.h>
21 #include <dw.h>
22 #include <dwmenu.h>
23 #include <dwsystem.h>
24 #include <color.h>
25 #include <d4base.h>
26
27 // custom header files
28 #include "sysdefs.h"                    // general typedefs and defs for procauto.c & comprom.c
29 #include "dbutil.h" // database access fns (only used by procauto.c & comprom.c presently)
30 #include "miscutil.h" // miscellaneous fns (only used by procauto.c & comprom.c presently)
31 #include "procutil.h"
32 #include "rmrsutil.h"
33 int DoCompromise(long, REQLSTNODE *);
34
35 #define RMRSERROR   -99
36 #define NOFILES -1
37 #define FOUND 0
38 #define NOTFOUND 1
39
40
41 // Assume all databases exist
42 // Handle for empty case
43
44 // SETUP RMRS SYSTEM DIRECTORY VARIABLES
45 // TO BE DEFINED IN "INIT/SETUP" FUNCTION
46 #ifdef RMRSSYSTEMDIRS
47 extern char pdesinputdirpath;
48 extern char pdesarchvdirpath;
49 #else
50     char pdesinputdirpath[50] = "";
51     char pdesarchvdirpath[50] = "archive\\";
52 #endif
53
54 extern HWND win;                        // Process | Auto mode comand window handle
55
56 int LoadSingFacFiles(char *);
57 long GetFacnum(char *);
58 long GetNxtAvlRepid(void);
59 int LoadEqpReqs(char *, long, long, int);
60 int LoadMatReqs(char *, long);
61 int LoadRepInfo(char *, long);
62 int TryResAlloc(void);
63 int AddRepRec(long, long, int);
64 int AddEqpRec(EQPREQREC);
65 int AddMatRec(char *, float, char *, long, char);
66 int AddRepInfoRec(REPINFOREC *);
67 void ArchiveFile(char *, char *, long);
68 long HrToSec(float);
69 long ChkOvrdPoss(long, float, long);
70 int AnyConflicts(REQLSTNODE *);
71 int SkipLines(FILE *, int, char *);
72 void ParseLine(char *, char *);
73 void PrintChkLst(REQLSTNODE *);
74 void ClearChkLst(REQLSTNODE *);
```

```
 75 int AllocMatls(REQLSTNODE *, long);
 76 int setrepairstatus(int, long);
 77 int GetRepairPrty(long);
 78 void cleanwin(void);
 79
 80 #ifdef RMRSPROCSTATS
 81 // processing info variable declarations
 82 typedef struct faclistnode {
 83    char *facnumstr[10];      // facility number string (e.g. B4058)
 84    int  numofreps;           // number of repairs in that submission
 85    struct faclistnode *next;
 86 } FACLISTNODE;
 87 int numofrepsprocd;     // # of repairs processed
 88 int numofconflreps;     // # of repairs w/ conflicts
 89 FACLISTNODE *faclist;  // LL of facilities processed
 90 #endif
 91
 92 // local statistic var
 93 static int fileloadcnt;
 94
 95 /*
 96  * int DoProcessAuto()
 97  *
 98  * ARGUMENTS: none
 99  *
100  * DESCRIPTION
101  *  Loads ALL new PDES facility repair files and checks for resource
102  *  availability conflicts, allocating material resources to a repair
103  *  if no conflicts arise or upon succesful resolution of those conflicts
104  *  Repairs go to one of three states: (P)ossible
105  *                                     (S)uspended
106  *                                     (X)canceled
107  *
108  * RETURNS
109  *
110  * AUTHOR
111  *  Applied Research Associates, Inc.
112  *
113  * MODIFICATIONS
114  *
115  */
116
117 int DoProcessAuto(void)
118 {
119     struct ffblk fblkptr;
120     int done;
121     char facnumstr[10];
122     char pdesinputfilestr[65];
123
124     // compose custom colors
125 {
125         int attr;
126         // colors used in COMPROMISE ENVIRONMENT
127         attr = bldattr(LTGREY, RED);
128         tblattr(SELBLINK, attr | GINTENSITY | GBL NK);
129         attr = bldattr(LTGREY, RED);
130         tblattr(SELNORM, attr | GINTENSITY);
131         attr = bldattr(BLACK, LTGREY);
132         tblattr(SELQTYBLINK, attr | GBLINK);
133         tblattr(SELQTY, attr);
134         attr = bldattr(WHITE, BLUE);
135         tblattr(HILITE, attr);
136
137         // colors used in miscellaneous MESSAGE BOXES
138         attr = bldattr(WHITE, LTGREY);
139         tblattr(GREYBOX, attr | GINTENSITY);
140         attr = bldattr(BLACK, LTGREY);
141         tblattr(GRYBTXT, attr);
142         tblattr(GRYBTXTBLINK, attr | GBLINK);
143         attr = bldattr(CYAN, BLUE);
144         tblattr(KEYALERT, attr | GBLINK);
145         attr = bldattr(CYAN, BLUE);
146         tblattr(KEYMESS, attr);
147         attr = bldattr(BLACK, GREEN);
```

```
148         tblattr(FLSHCHR, attr | GBLINK);
149     }
150     // open required databases (old method)
151     repair_dbf = d4use_excl("REPAIR.DBF");
152     matreq_dbf = d4use_excl("MATREQ.DBF");
153     eqpreq_dbf = d4use_excl("EQPREQ.DBF");
154     eqpsup_dbf = d4use_excl("EQPSUP.DBF");
155     matsup_dbf = d4use_excl("MATSUP.DBF");
156     repinfo_dbf = d4use_excl("REPINFO.DBF");
157
158     fileloadcnt = 0;                    // init file load count
159     // RETRIEVE NEW PDES FILES FROM INPUT DIRECTORY
160     // Since files are removed from this directory after processing,
161     // any .EQP,.MAT,.OUT files found there are considered to be new.
162     strcpy(pdesinputfilestr, pdesinputdirpath);
163     strcat(pdesinputfilestr, "*.EQP");
164     done = findfirst(pdesinputfilestr, & fblkptr, 0);
165     if (done == NOFILES)
165     {
166         vatputs(win, 1, 2, "WARNING: NO FILES FOUND IN              ");
167         vatputs(win, 2, 2, "         SPECIFIED DIRECTORY            ");
168         vatputs(win, 4, 2, "** Check PDES Input directory setting   ");
169         vatputs(win, 6, 5, "-> press any key to continue            ");
170         getkey();
171         return (0);
172     }
173     // Load facility repair file sets one at a time
174     while (done != NOFILES)
174     {
175         strcpy(facnumstr, fblkptr.ff_name);
176         facnumstr[strlen(facnumstr) - 4] = '\0';
177         vatputf(win, 2, 2, "Loading facility files for:  %5s ", facnumstr);
178         modattr(win, 2, 30, 7, HELP);
179
180         LoadSingFacFiles(facnumstr);
181
182         done = findnext( & fblkptr);
183     }
184     d4close_all();                      //
185     // reindex files that changed
186     open_matreq(1);                     // 3 files open
187     open_eqpreq(1);                     // 6 files open
188     open_repair(1);                     // 3 files open
189     open_repinfo(1);                    // 2 files open
190     d4close_all();                      //
191     // reopen files to use for resource allocation
192     repair_dbf = d4use_excl("REPAIR.DBF");
193     matreq_dbf = d4use_excl("MATREQ.DBF");
194     eqpreq_dbf = d4use_excl("EQPREQ.DBF");
195     eqpsup_dbf = d4use_excl("EQPSUP.DBF");
196     matsup_dbf = d4use_excl("MATSUP.DBF");
197
198     cleanwin();
199     TryResAlloc();
200
201     d4close_all();
202     open_matreq(1);                     // 3 files open
203     open_eqpreq(1);                     // 6 files open
204     open_repair(1);                     // 3 files open
205     d4close_all();
206     return (1);
207
208 }
209
210 int LoadSingFacFiles(char * facnumstrptr)
211 /*
212  * ARGUMENT
213  *   (char *)  facnumstrptr - pointer to facility number string (e.g. "B4058")
214  *
215  * DESCRIPTION
216  *   Loads the repair-specific databases (REPAIR.DBF, EQPREQ.DBF, MATREQ.DBF,
217  *   and REPINFO.DBF) with the repair data from a single facility repair file
218  *   set (i.e. the .EQP, .MAT, and .OUT files) specified by 'facnumstrptr'
219  *
```

```
220  * RETURNS
221  *   (int) status code: SUCCESS or FAILURE
222  *
223  * DATABASES AFFECTED (whether direct or indirect)
224  *    facprt_dbf (facprt_ndx1 - FACNUM index) - d4seek used
225  *    repair_dbf (no index used)
226  *    eqpreq_dbf (no index used)
227  *    matreq_dbf (no index used)
228  *    repinfo_dbf (no index used)
229  *
230  */
231 {
232     long facnum = 0L, indxrepid = 0L, currrepid = 0L;
233     int priority = 0, err = 0;
234
235     currrepid = GetNxtAvlRepid();       // get next avail repair id
236     indxrepid = currrepid;
237     facnum = GetFacnum(facnumstrptr);
238     // look up facility priority in FACPRTY.DBF
239     if (! (priority = GetFacPrty(facnum)))
239     {
240         u4error(-1, "WARNING: Facility not listed in",
241             "FACPRTY.DBF",
242             ".. processing will be aborted",
243             "for this facility", (char *) 0);
244         return (FAILURE);               // abort
245     }
246     err = LoadEqpReqs(facnumstrptr, indxrepid, facnum, priority);
247     if (! err) err = LoadMatReqs(facnumstrptr, indxrepid);
248     if (! err) err = LoadRepInfo(facnumstrptr, indxrepid);
249     return (err);
250 }
251
252 int LoadEqpReqs(char * facnumstrptr, long indxrepid, long facnum, int priority)
252 {
253 /*
254  * ARGUMENT
255  *   (char *)  facnumstrptr - pointer to facility number string (e.g. "B4058")
256  *   (long)    indxrepid    - index repid
257  *   (long)    facnum       - facility number
258  *   (int)     priority     - AB mission-critical facility priority assignment
259  *
260  * DESCRIPTION
261  *   Loads the repair equipment requirements into EQPREQ.DBF from the .EQP
262  *   file specified by 'facnumstrptr'; also, because each repair is
263  *   guaranteed to have at least 1 equipment requirement, namely a "repair
264  *   team", this fact is exploited within this function to produce a repair
265  *   data record (REPAIR.DBF) for each repair
266  *
267  * RETURNS
268  *   (int) error status code:   < 0 - error
269  *                              = 0 - success
270  *
271  * DATABASES AFFECTED (whether direct or indirect)
272  *    eqpreq_dbf (no index used)
273  *
274  */
275     char eqpfilename[50];
276     FILE * eqpfileptr;
277     long currrepid;
278     char prev[6], * currptr;
279     char fileline[80];
280     int eqpqty;
281     int err;
282     EQPREQREC eqprecbuf;
283
284     // load equipment requirements from .EQP file into EQPREQ.DBF *
285     strcpy(eqpfilename, pdesinputdirpath);
286     strcat(eqpfilename, facnumstrptr);
287     strcat(eqpfilename, ".EQP");
288     if ((eqpfileptr = fopen(eqpfilename, "r")) != NULL)
288     {
289         currrepid = indxrepid;
290         AddRepRec(facnum, currrepid, priority);
```

```
291         strcpy(prev, "1");
292         fgets(fileline, 80, eqpfileptr);
293         strtok(fileline, "\"");
294         currptr = strtok(NULL, "\"");
295
296         while (! feof(eqpfileptr))
296         {
297             // check to see this is eqp for diff repair
298             // if yes, create new repair record with unique id #
299             if (atoi(currptr) > atoi(prev))
299             {
300                 strcpy(prev, currptr);
301                 currrepid = indxrepid + (atoi(currptr) - 1);
302                 err = AddRepRec(facnum, currrepid, priority);
303             }
304
305             // initialize eqp record structure
306             eqprecbuf.eqpid = 0L;
307             strcpy(eqprecbuf.eqpdescstr, "");
308             eqprecbuf.start = 0L;
309             eqprecbuf.duration = 0L;
310             eqprecbuf.repid = currrepid;
311             eqprecbuf.status = 'N';
312
313             // parse equipment record for information
314             strtok(NULL, "\"");
315             strcpy(eqprecbuf.eqpdescstr, strtok(NULL, "\""));
316             strtok(NULL, "\"");
317             eqpqty = atoi(strtok(NULL, "\""));
318             strtok(NULL, "\""); strtok(NULL, "\""); strtok(NULL, "\"");
319             eqprecbuf.duration = HrToSec(atof(strtok(NULL, "\'")));
320
321             // add separate eqpuipment requirment record
322             //  for each individual piece of equipment
323             for (; eqpqty; eqpqty--)
324                 err = AddEqpRec(eqprecbuf);
325             SkipLines(eqpfileptr, 1, fileline);
326             strtok(fileline, "\"");
327             currptr = strtok(NULL, "\"");
328         }
329         fclose(eqpfileptr);
330         ArchiveFile(eqpfilename, ".EQP", indxrepid);
331         return (RMRSSUCCESS);
332     }
333     else
333     {
334         u4error(-1, "FILE ACCESS RMRSERROR!", "can't .EQP open file:", eqpfilename, (char *) 0)
;
335         return (-1);
336     }
337 }
338
339
340 int LoadMatReqs(char * facnumstrptr, long indxrepid)
340 {
341 /*
342  * ARGUMENT
343  *   (char *)  facnumstrptr - pointer to facility number string (e.g. "B4058")
344  *   (long)    indxrepid    - index repid
345  *
346  * DESCRIPTION
347  *   Loads the repair material requirements into MATREQ.DBF from the .MAT
348  *   file specified by 'facnumstrptr'
349  *
350  * RETURNS
351  *   (int) error status code:   < 0 - error
352  *                              = 0 - success
353  *
354  * DATABASES AFFECTED (whether direct or indirect)
355  *    matreq_dbf (no index used)
356  *
357  */
358
359     char matfilename[50];
```

```
360     FILE * matfileptr;
361     long currrepid;
362     char * currptr;
363     char fileline[80];
364     char * matdescptr, * matunitptr;
365     float matqty;
366     int err;
367
368     // load material requirements from .MAT file into MATREQ.DBF **
369     strcpy(matfilename, pdesinputdirpath);
370     strcat(matfilename, facnumstrptr);
371     strcat(matfilename, ".MAT");
372     if ((matfileptr = fopen(matfilename, "r")) != NULL)
372     {
373         fgets(fileline, 80, matfileptr);
374         strtok(fileline, "\"");
375         currptr = strtok(NULL, "\"");
376         currrepid = indxrepid + (atoi(currptr) - 1);
377         while (! feof(matfileptr))
377         {
378             strtok(NULL, "\"");
379             matdescptr = strtok(NULL, "\"");
380             strtok(NULL, "\"");
381             matqty = atof(strtok(NULL, "\""));
382             strtok(NULL, "\"");
383             matunitptr = strtok(NULL, "\"");
384
385             err = AddMatRec(matdescptr, matqty, matunitptr, currrepid, 'N');
386
387
388             fgets(fileline, 80, matfileptr);
389             strtok(fileline, "\"");
390             currptr = strtok(NULL, "\"");
391             currrepid = indxrepid + (atoi(currptr) - 1);
392         }
393
394         fclose(matfileptr);
395         ArchiveFile(matfilename, ".MAT", indxrepid);
396         return (RMRSSUCCESS);
397     }
398     else
398     {
399         u4error(-2, "FILE ACCESS RMRSERROR!",
400             "can't open .MAT file:", matfilename, (char *) 0);
401         return (-2);
402     }
403 }
404
405
406 int LoadRepInfo(char * facnumstrptr, long indxrepid)
406 {
407 /*
408  * ARGUMENT
409  *   (char *)  facnumstrptr - pointer to facility number string (e.g. "B4058")
410  *   (long)    indxrepid    - index repid
411  *
412  * DESCRIPTION
413  *   Loads the repair information into REPINFO.DBF from the .OUT
414  *   file specified by 'facnumstrptr'
415  *
416  * RETURNS
417  *   (int) error status code:   < 0 - error
418  *                              = 0 - success
419  *
420  * DATABASES AFFECTED (whether direct or indirect)
421  *    repinfo_dbf (no index used)
422  *
423  */
424
425     REPINFOREC buffer;
426     char outfilename[50];
427     FILE * outfileptr;
428     char linestr[80];
429     char elenumstr[50];
```

```
430     char remstr[80];
431     int cnt = 0, err;
432
433     // load facility repair info from .OUT file into REPINFO.DBF **
434     strcpy(outfilename, pdesinputdirpath);//  build
435     strcat(outfilename, facnumstrptr); //   filename
436     strcat(outfilename, ".OUT");       //    string
437     if ((outfileptr = fopen(outfilename, "r")) != NULL)
437     {
438         // get facility header information
439         SkipLines(outfileptr, 1, linestr);
440         while (linestr[0] != '_')
441             SkipLines(outfileptr, 1, linestr);
442         SkipLines(outfileptr, 6, linestr);
443         ParseLine(linestr, buffer.facfunc);
444         SkipLines(outfileptr, 2, linestr);
445         ParseLine(linestr, buffer.facdesc);
446         SkipLines(outfileptr, 8, linestr);
447         // loop through each repair report
448         while (! feof(outfileptr))
448         {
449             buffer.repid = indxrepid + cnt++;
450             ParseLine(linestr, elenumstr);
451             buffer.elenum = atoi(elenumstr);
452             SkipLines(outfileptr, 1, linestr);
453             ParseLine(linestr, buffer.eledesc);
454             SkipLines(outfileptr, 1, linestr);
455             ParseLine(linestr, buffer.dammode);
456             SkipLines(outfileptr, 1, linestr);
457             strcpy(buffer.damw, "x");
458             strcpy(buffer.daml, "x");
459             strcpy(buffer.damh, "x");
460             // compensate for TAB
461             while ((linestr[9] == 'D') || (linestr[2] == 'D'))
461             {
462                 // check to see which data var to load
463                 switch (linestr[16])
463                 {
464                 case ('W') :
465                     ParseLine(linestr, buffer.damw);
466                     break;
467                 case ('L') :
468                     ParseLine(linestr, buffer.daml);
469                     break;
470                 case ('H') :
471                     ParseLine(linestr, buffer.damh);
472                     break;
473                 default :;
474                 }
475                 // handle case in which TAB spacing is used
476                 switch (linestr[9])
476                 {
477                 case ('W') :
478                     ParseLine(linestr, buffer.damw);
479                     break;
480                 case ('L') :
481                     ParseLine(linestr, buffer.daml);
482                     break;
483                 case ('H') :
484                     ParseLine(linestr, buffer.damh);
485                     break;
486                 default :;
487                 }
488                 SkipLines(outfileptr, 1, linestr);
489             }
490             ParseLine(linestr, buffer.repstgy);
491             SkipLines(outfileptr, 1, linestr);
492
493             // retrieve remarks entry
494             // locate "  4.) Remarks :  " line in .OUT file
495             while ((atoi(strtok(linestr, "."))) != 4)
496                 SkipLines(outfileptr, 1, linestr);
497             SkipLines(outfileptr, 2, linestr);
498
```

```
499             strcpy(buffer.remline, linestr);
500             trimstr(buffer.remline);
501             strcpy(remstr, "\0");
502             // if non-empty remarks section, build text lines
503             // into one continuous string -> "remstr"
504             while (strlen(buffer.remline) > 1)
505             {
506                 strcat(remstr, buffer.remline);
507                 strcpy(buffer.remline, "\n");
508                 SkipLines(outfileptr, 1, linestr);
509                 strcpy(buffer.remline, linestr);
510                 trimstr(buffer.remline);
511             }
512 //   (CSS)       { long *memofilestats;
513 //         memofilestats=m4check(f4ref("REMARKS"));
514 //         // REMARKS LOADING IS NOT OPERATIONAL: will fix soon
515 //         { int i;
516 //           for(i=0;i<=5;i++)
517 //             vatputf(win,4,1+(6*i),"<%ld>",*(memofilestats+(i*sizeof(long))));
518 //         }
519 //       }
520
521             err = AddRepInfoRec(& buffer);
522
523             while ((linestr[0] != '_') && (! feof(outfileptr)))
524                 SkipLines(outfileptr, 1, linestr);
525             if (! feof(outfileptr))
526                 SkipLines(outfileptr, 8, linestr);
527         }
528         fclose(outfileptr);
529         ArchiveFile(outfilename, ".OUT", indxrepid);
530         return (RMRSSUCCESS);
531     }
531     else
532     {
532         u4error(-3, "FILE ACCESS RMRSERROR!", "can't open .OUT file:", outfilename, (char *) 0)
;
533         return (-3);
534     }
535 }
536
537 void ArchiveFile(char * oldfilename, char * fileext, long indxrepid)
537 {
538 /*
539  * ARGUMENT
540  *   (char *)   oldfilename - present path-specified filename
541  *   (char *)   fileext     - filename extension
542  *   (long)     indxrepid   - index repair id  (unique for each file set)
543  *
544  * DESCRIPTION
545  *   Transfers a RMRS-loaded file from the PDES Input directory to
546  *   the PDES Archive directory; the new file gets the prefix "ARF"
547  *   (for Archived Repair File) followed by the reference repair id number
548  *   (left zero-padded to 5 digits) corresponding to the first repair in the
549  *   file set; for example, if the file set designated B4058 was loaded, and
550  *   the first repair from that set was mapped to repair id #303, then the
551  *   corresponding archived files would be named: ARF00303.EQP, ARF00303.MAT,
552  *   and ARF00303.OUT
553  *
554  * RETURNS
555  *
556  * NOTES
557  *   (int) interrupt 56H status codes : 3 - Path not found
558  *                                      5 - Access denied
559  *                                     17 - Path not found
560  *
561  * DATABASES AFFECTED (whether direct or indirect)
562  *   none
563  *
564  */
565     char archivefilename[50], str[25], errmess[50];
566     int status, len, i;
567
568     vatputf(win, 4, 2 + fileloadcnt++, "*");
```

```
569     strcpy(archivefilename, pdesarchvdirpath);
570     strcat(archivefilename, "ARF");
571
572     // convert indxrepid to ascii and pad left with zeros
573     ltoa(indxrepid, str, 10);
574     len = strlen(str);
575     for (i = 1; i <= len; i++)
576         str[5 - i] = str[len - i];
577     for (i = 0; i <= (4 - len); i++)
578         str[i] = '0';
579     str[5] = '\0';
580
581     strcat(archivefilename, str);
582     strcat(archivefilename, fileext);
583     status = move_file(oldfilename, archivefilename);
584     if (status)
584     {
585         switch (status)
585         {
586         case (3) :
587         case (17) : strcpy(errmess, "Path not found");
588         case (5) : strcpy(errmess, "Access denied");
589         default : strcpy(errmess, "Undefined error");
590         }
591         u4error(status, "File archive error"
592             , errmess
593             , " "
594             , oldfilename
595             , archivefilename, (char *) 0);
596     }
597     return;
598 }
599
600
601 int AddEqpRec(EQPREQREC eqprecbuf)
601 {
602 /*
603  * ARGUMENT
604  *   (EQPREQREC)  eqprecbuf - equipment requirement record buffer
605  *
606  * DESCRIPTION
607  *   Populates an equipment requirement record and appends it to the
608  *   EQPREQ.DBF database
609  *
610  * RETURNS
611  *   (int) d4append() return status code
612  *
613  * DATABASES AFFECTED (whether direct or indirect)
614  *    eqpreq_dbf (no index used)
615  *
616  */
617     d4select(eqpreq_dbf);
618     f4r_long(f4ref("EQPID"), 0L);
619     f4r_str(f4ref("EQPDESC"), eqprecbuf.eqpdescstr);
620     f4r_long(f4ref("START"), 0L);
621     f4r_long(f4ref("DURATION"), eqprecbuf.duration);
622     f4r_long(f4ref("REPID"), eqprecbuf.repid);
623     f4r_char(f4ref("STATUS"), eqprecbuf.status);
624     return (d4append());
625 }
626
627
628 int AddMatRec(char * matdescptr, float matqty, char * matunitptr, long repid, char status)
628 {
629 /*
630  * ARGUMENT
631  *   (MATREQREC)  matrecbuf - material requirement record buffer
632  *
633  * DESCRIPTION
634  *   Populates a material requirement record and appends it to the
635  *   MATREQ.DBF database
636  *
637  * RETURNS
638  *   (int) d4append() return status code
```

```
639  *
640  * DATABASES AFFECTED (whether direct or indirect)
641  *    matreq_dbf (no index used)
642  *
643  */
644
645     // add material requirement record to MATREQ.DBF
646     d4select(matreq_dbf);
647     f4r_long(f4ref("MATID"), 0);
648     f4r_str(f4ref("MATDESC"), matdescptr);
649     f4r_double(f4ref("QTY"), matqty);
650     f4r_str(f4ref("UNIT"), matunitptr);
651     f4r_long(f4ref("REPID"), repid);
652     f4r_char(f4ref("STATUS"), status);
653     return (d4append());
654 }
655
656
657 int AddRepInfoRec(REPINFOREC * rec)
657 {
658 /*
659  * ARGUMENT
660  *   (REPINFOREC)  repinforecbuf - repair information record buffer
661  *
662  * DESCRIPTION
663  *   Populates a repair information record and appends it to the
664  *   REPINFO.DBF database
665  *
666  * RETURNS
667  *   (int) d4append() return status code
668  *
669  * DATABASES AFFECTED (whether direct or indirect)
670  *    repinfo_dbf (no index used)
671  *
672  */
673
674     // add repair .OUT file info record to REPINFO.DBF
675     d4select(repinfo_dbf);
676     f4r_long(f4ref("REPID"), rec->repid);
677     f4r_str(f4ref("FACFUNC"), rec->facfunc);
678     f4r_str(f4ref("FACDESC"), rec->facdesc);
679     f4r_int(f4ref("ELENUM"), rec->elenum);
680     f4r_str(f4ref("ELEDESC"), rec->eledesc);
681     f4r_str(f4ref("DAMMODE"), rec->dammode);
682     f4r_str(f4ref("DAMW"), rec->damw);
683     f4r_str(f4ref("DAML"), rec->daml);
684     f4r_str(f4ref("DAMH"), rec->damh);
685     f4r_str(f4ref("REPSTGY"), rec->repstgy);
686 #ifdef DEBUG
687 (CS$)  printf("\nnumber written >%d<",m4write(f4ref("REMARKS"),d4recno(), rec->remline,80));
688 #endif
689
690     return (d4append());
691 }
692
693 int SkipLines(FILE * fptr, int numoflines, char * returnline)
693 {
694 /*
695  * ARGUMENT
696  *   (FILE *) fptr       - file pointer
697  *   (int)    numoflines - number of lines to skip
698  *   (char*)  returnline - string containing current line of file
699  *
700  * DESCRIPTION
701  *   Skips the specified number of lines in the specified file and
702  *   returns the current line
703  *
704  * RETURNS
705  *   (int) status code
706  *
707  * DATABASES AFFECTED
708  *   none
709  *
710  */
```

```
711     int i;
712
713     for (i = 1; i <= numoflines; i++)
714         fgets(returnline, 80, fptr);
715     return (1);
716 }
717
718 void ParseLine(char * linestr, char * returnstr)
718 {
719 /*
720  * ARGUMENT
721  *   (char *) linestr   - line to be parsed
722  *   (char *) returnstr - return string
723  *
724  * DESCRIPTION
725  *   Parses data lines within .OUT text file
726  *
727  * RETURNS
728  *
729  * DATABASES AFFECTED
730  *   none
731  *
732  */
733     strtok(linestr, ":");
734     strcpy(returnstr, strtok(NULL, "\n"));
735 }
736
737 long HrToSec(float hours)
737 {
738 /*
739  * ARGUMENT
740  *   (float)  hours - hours value to convert
741  *
742  * DESCRIPTION
743  *   Converts hours to seconds
744  *
745  * RETURNS
746  *   (long)   number of seconds
747  *
748  */
749     return ((long) (hours * 3600.0));
750 }
751
752 int TryResAlloc(void)
752 {
753 /*
754  * ARGUMENT
755  *
756  * DESCRIPTION
757  *   Prioritizes (N)ew and (O)verriden repairs, analyzes overall
758  *   resource availability for each repair, and acts appropriately
759  *   according to the results of the analysis.  Thus, materials are
760  *   allocated to the repair if no conflicts persist.
761  *
762  *   Equipment conflicts result in (X)cancel-ation of repair.
763  *   Compromise mode is available for a repair with only material
764  *   conflicts, otherwise those repair is (S)uspended.
765  *
766  * RETURNS
767  *   (int) status code
768  *
769  * DATABASES AFFECTED (whether direct or indirect)
770  *    repair_dbf (repair_ndx1 - REPID index       ) - d4seek used
771  *               (repair_ndx2 - prioritized index ) - d4skip used
772  *    eqpsup_dbf (eqpsup_ndx1 - EQPDESC index     ) - d4seek used
773  *    eqpreq_dbf (eqpreq_ndx1 - REPID index       ) - d4seek, d4skip used
774  *    matsup_dbf (matsup_ndx1 - MATDESC index     ) - d4seek used
775  *               (matsup_ndx2 - MATID index       ) - (CS$) to be added
776  *    matreq_dbf (matreq_ndx1 - REPID index       ) - d4seek, d4skip used
777  *    matreq_dbf (matreq_ndx2 - MATID index       ) - d4seek, d4skip used
778  *
779  */
780
781     long curr;
```

```
782     char eqpdescstr[26], matdescstr[26], repstatus;
783     REQLSTNODE * reqchklst, * lastnode;
784     float matqtyneeded, matqtyavail;
785     long matid, ovrdrecno;
786     int status;
787     CONTEXT def1, def2;
788
789     vatputs(win, 2, 2, "Allocating material resources to repairs...");
790
791
792     // Index REPAIR.DBF for searching on "REPID"
793     // Used in GetRepairPrty() to retrieve priority
794     d4select(repair_dbf);
795     repair_ndx1 = i4open("REPRNDX1");
796     if (repair_ndx1 < 0)
796     {
797         u4error(-99, "repair_ndx1",(char *) 0);
798     }
799     // Index REPAIR.DBF for sequencing through (N)ew repairs
800     repair_ndx2 = i4open("REPRNDX2");
801     if (repair_ndx2 < 0)
801     {
802         u4error(-99, "repair_ndx2",(char *) 0);
803     }
804
805     // index EQPSUPP.DBF for searching on "EQPDESC"
806     // used for locating equipment requirement pieces in Air Base supply
807     d4select(eqpsup_dbf);
808     eqpsup_ndx1 = i4open("EQSUNDX1");
809     if (eqpsup_ndx1 < 0)
809     {
810         u4error(-99, "eqpsup_ndx1",(char *) 0);
811     }
812
813     // index EQPREQ.DBF for searching on "REPID"
814     // used to find all equipment requirements for a repair
815     d4select(eqpreq_dbf);
816     eqpreq_ndx1 = i4open("EQRENDX1");
817     if (eqpreq_ndx1 < 0)
817     {
818         u4error(-99, "eqpreq_ndx1",(char *) 0);
819     }
820
821     // index MATSUPP.DBF for searching on "MATDESC"
822     // used for locating material requirement pieces in Air Base supply
823     d4select(matsup_dbf);
824     matsup_ndx1 = i4open("MASUNDX1");
825     if (matsup_ndx1 < 0)
825     {
826         u4error(-99, "matsup_ndx1",(char *) 0);
827     }
828     // NOTICE: this index file must be added (CS 12/30)
829     // index MATSUPP.DBF for searching on "MATID"
830     // used for in AllocMats() for for finding material listing in supply
831 //   matsup_ndx2 = i4open("MASUNDX2");
832 //   if (matsup_ndx2 <0) {
833 //       u4error(-99,"matsup_ndx2",(char *)0);
834 //   }
835
836     // index MATREQ.DBF for searching on "REPID"
837     // used to find all material requirements for a repair
838     d4select(matreq_dbf);
839     matreq_ndx1 = i4open("MARENDX1");
840     if (matreq_ndx1 < 0)
840     {
841         u4error(-99, "matreq_ndx1",(char *) 0);
842     }
843     // index MATREQ.DBF for searching on "MATID"
844     // used to search all instances of a particular material type
845     // for priority-override candidates
846     matreq_ndx2 = i4open("MARENDX2");
847     if (matreq_ndx2 < 0)
847     {
848         u4error(-99, "matreq_ndx2",(char *) 0);
```

```
849     }
850
851     // create and init data structure for resource requirements check list
852     reqchklst = (REQLSTNODE *) malloc(sizeof (REQLSTNODE));
853     reqchklst->type = 'H';
854     reqchklst->recno = 0;
855     reqchklst->availstatus = 'X';
856     reqchklst->next = (REQLSTNODE *) NULL;
857     lastnode = reqchklst;
858
859     // PROCESS ALL (N)EW REPAIRS
860     d4select(repair_dbf);
861     d4top();
862     repstatus = f4char(f4ref("STATUS"));
863 //   x4list();
864 //   getkey();
865 //   exit(0);
866 //   while (((repstatus=f4char(f4ref("STATUS"))) == 'N') || (repstatus == 'O')) {
867     while ((repstatus = f4char(f4ref("STATUS"))) == 'N')
867     {
868         curr = f4long(f4ref("REPID"));
869
870         // CHECK EQP REQUIREMENTS FOR AVAILABILITY
871         d4select(eqpreq_dbf);
872         i4select(eqpreq_ndx1);
873         if (d4seek_double(curr) == FOUND)
873         {
874             while ((f4long(f4ref("REPID"))) == curr)
874             {
875
876                 // create check list node for eqp req
877                 lastnode->next = (REQLSTNODE *) malloc(sizeof (REQLSTNODE));
878                 lastnode = lastnode->next;
879                 lastnode->type = 'E';
880                 lastnode->recno = d4recno();
881                 lastnode->availstatus = 'x';
882                 lastnode->resrecno = 0;
883                 lastnode->resqty = 0.0;
884                 lastnode->ovrd = FALSE;
885                 lastnode->next = (REQLSTNODE *) NULL;
886
887                 // search EQPSUPP.DBF for eqp piece
888                 strcpy(eqpdescstr, f4str(f4ref("EQPDESC")));
889                 d4select(eqpsup_dbf);
890                 i4select(eqpsup_ndx1);
891                 uppercase(eqpdescstr);
892                 status = d4seek_str(eqpdescstr);
893                 if (status == FOUND)
893                 {
894                     //  leave marked as (N)ew (in EQPREQ.DBF)
895                     lastnode->availstatus = 'F';
896                 }
897                 else
897                 {
898                     // mark as (M)issing (in EQPREQ.DBF)
899                     lastnode->availstatus = 'M';
900                     d4select(eqpreq_dbf);
901                     f4r_char(f4ref("STATUS"), 'M');
902                 }
903                 d4select(eqpreq_dbf);
904                 d4skip(1);
905             }
906         }
907
908         // CHECK MAT REQUIREMENTS FOR AVAILABILITY
909         d4select(matreq_dbf);
910         i4select(matreq_ndx1);
911         if (d4seek_double(curr) == FOUND)
911         {
912             while ((f4long(f4ref("REPID"))) == curr)
912             {
913                 lastnode->next = (REQLSTNODE *) malloc(sizeof (REQLSTNODE));
914                 lastnode = lastnode->next;
915                 lastnode->type = 'M';
```

```
916                 lastnode->recno = d4recno();
917                 lastnode->availstatus = 'x';
918                 lastnode->resrecno = 0;
919                 lastnode->resqty = 0.0;
920                 lastnode->ovrd = FALSE;
921                 lastnode->next = (REQLSTNODE *) NULL;
922                 matqtyneeded = f4double(f4ref("QTY"));
923                 // search MATSUPP.DBF for material
924                 strcpy(matdescstr, f4str(f4ref("MATDESC")));
925                 d4select(matsup_dbf);
926                 i4select(matsup_ndx1);
927                 uppercase(matdescstr);
928                 status = d4seek_str(matdescstr);
929                 if (!status == FOUND)
929                 {
930                     matid = f4long(f4ref("MATID"));
931                     matqtyavail = f4double(f4ref("QTY"));
932                     d4select(matreq_dbf);
933                     f4r_long(f4ref("MATID"), matid);
934                     d4write(d4recno());
935                     if (matqtyavail >= matqtyneeded)
935                     {
936                         lastnode->availstatus = 'E';
937                     }
938                     else
938                     {
939                         d4select(repair_dbf);
940                         context_save(& def1);
941                         d4select(matreq_dbf);
942                         context_save(& def2);
943                         /* (I)nadequate supply to meet needs, check override poss */
944                         ovrdrecno = ChkOvrdPoss(matid, (matqtyneeded - matqtyavail), curr);
945                         context_restore(& def1);
946                         context_restore(& def2);
947                         if (ovrdrecno)
947                         {
948                             // (O)verride is possible, save record number
949                             lastnode->availstatus = 'O';
950                             lastnode->resrecno = ovrdrecno;
951                         }
952                         // (I)nadequate quantity in AB material supply
953                         else
954                             lastnode->availstatus = 'I';
955                     }
956                 }
957                 // matching entry not found in AB mat supply: (M)issing
958                 else
958                 {
959                     lastnode->availstatus = 'M';
960                 }
961                 d4select(matreq_dbf);
962                 d4skip(1);
963             }
964         }
965         else
965         {
966             // no materials associated with repair
967         }
968
969
970         // see if any resource conflicts arose
971         if (AnyConflicts(reqchklst))
971         {
972             /* check to see if one was an equipment conflict */
973             if (iseqpconf(reqchklst))
973             {
974                 /* notify user compromise not possible & cancel repair */
975 //              alerttoeqpconf();   // (CS$) to be added
976                 setrepairstatus(XCANCEL, curr);
977             }
978             else
978             {
979                 /* ask user if he would like to attempt a compromise */
980                 /* to resolve material conflicts                     */
```

```
                if (askcomp())
                {
                    d4select(repair_dbf);
                    i4select(repair_ndx2);
                    context_save(& def1);
                    if (DoCompromise(curr, reqchklst) == SUCCESS)
                    {
                        AllocMatls(reqchklst, curr);
                        /* leave eqp reqs marked (N)ew */
                    }
                    else
                    {
                        /* unsuccesful compromise attempt */
                        setrepairstatus(SUSPEND, curr);
                    }
                    context_restore(& def1);
                }
                else
                {
                    /* user chose not to attempt compromise at this time */
                    setrepairstatus(SUSPEND, curr);
                }
            }
        }
        else
        {
            AllocMatls(reqchklst, curr);
            /* leave eqp reqs marked (N)ew */
        }
        // get next repair
        d4select(repair_dbf);
        d4top();
        d4top();                        // this redundant function call is not a mistake
        ClearChkLst(reqchklst);
        lastnode = reqchklst;
    }
    free((REQLSTNODE *) reqchklst);
    d4close_all();

    return (1);
}


long ChkOvrdPoss(long matid, float matqtyneeded, long repid)
{
/*
 * ARGUMENT
 *   (long)     matid          - RMRS system-assigned material id number
 *                               of mat resource under conflict
 *   (float)    matqtyneeded   - quantity of material needed by the conflicting
 *                               material requirement
 *   (long)     repid          - RMRS system assigned repair id of repair to
 *                               which the conflicting mat requirement belongs
 *
 * DESCRIPTION
 *  This function searches through the (A)llocated material records
 *  corresponding to 'matid' in an attempt to find the lowest priority
 *  repair that is holding the least qty of sufficient resources to
 *  statisfy the particular conflicting material need of the higher priority
 *  repair specified by 'repid'. If one is found, the corresponding record
 *  number is returned. Otherwise, zero is returned.
 *
 * RETURNS
 *   (long)  n - record number of priority override candidate
 *               NOTE: This is the record number of the material requirement
 *               record in the MATREQ.DBF from which the materials will
 *               be taken.
 *           0 - none found
 *
 * Databases affected
 *    repair_dbf (repair_ndx1 - REPID index) - d4seek used
 *    matreq_dbf (matreq_ndx2 - MATID index) - d4seek, d4skip used
 *
 */
```

```
1049      long same, selrecno = 0L, xrecno, xrepid, xmatid;
1050      int minprty, xprty, flgprty;
1051      char xstatus;
1052      float xqty, currminqty;
1053
1054      // Get priority of the conflict-owning repair
1055      flgprty = GetRepairPrty(repid);
1056      minprty = flgprty;
1057
1058      // MATREQ.DBF: indexed on "MATID"
1059      d4select(matreq_dbf);
1060      i4select(matreq_ndx2);
1061      // this index file groups like materials together, so find first
1062      // and skip sequentially through the list
1063      same = matid;
1064      d4seek_double(matid);
1065      d4recno();
1066      xmatid = f4long(f4ref("MATID"));
1067      // check all matching (A)llocated material records for
1068      //    quantity available and associated priority
1069      // NOTE: vars w/ prefix 'x' hold data belonging to current
1070      //       mat requirement under scrutiny
1071      do
1071      {
1072          xstatus = f4char(f4ref("STATUS"));
1073          xqty = f4double(f4ref("QTY"));
1074          xrepid = f4long(f4ref("REPID"));
1075          xrecno = d4recno();
1076          xprty = GetRepairPrty(xrepid);
1077          if (xstatus == 'A')
1077          {
1078              // does this mat req belong to a repair with a LOWER priority
1079              // (higher number) and does the it have at least the qty needed ??
1080              if ((xprty > minprty) && (xqty >= matqtyneeded))
1080              {
1081                  minprty = xprty;
1082                  currminqty = xqty;
1083                  selrecno = xrecno;
1084              }
1085              // does this mat req belong to a repair with the SAME priority
1086              // as the current minimum selected mat req and does the it have a
1087              // LOWER qty than the current minimum selected mat req
1088              else if ((xprty == minprty) && (xprty != flgprty) &&
1089                       (xqty >= matqtyneeded) && (xqty < currminqty))
1089              {
1090                  currminqty = xqty;
1091                  selrecno = xrecno;
1092              }
1093          }
1094          d4select(matreq_dbf);
1095          i4select(matreq_ndx2);
1096          d4skip(1);
1097          xmatid = f4long(f4ref("MATID"));
1098      }
1098      while ((xmatid == same) && (! d4eof()));
1099
1100      return (selrecno);            /* return subordinate matreq record number, 0 for none*/
1101  }
1102
1103  int AnyConflicts(REQLSTNODE * lstptr)
1103  {
1104  /*
1105   * ARGUMENT
1106   *   (REQLSTNODE *)  lstptr - pointer to resource availability analysis list
1107   *
1108   * DESCRIPTION
1109   *   Checks the completed resource availability analysis list for any
1110   *   conflicts.
1111   *
1112   *     Non-conflict statuses:  for equipment .... (F)ound
1113   *                             for material...... (E)xistent
1114   *
1115   *   note: this availability status refers to the codes used in the
1116   *   resource availability analysis linked list and not the STATUS fields
```

```
1117  *   used in the databases
1118  *
1119  * RETURNS
1120  *   (int) number of resource availability conflicts for the repair
1121  *
1122  * DATABASES AFFECTED
1123  *   none
1124  */
1125
1126     int numofconflicts;
1127
1128     numofconflicts = 0;
1129     while (lstptr->next != (REQLSTNODE *) NULL)
1129     {
1130         lstptr = lstptr->next;
1131         switch (lstptr->availstatus)
1131         {
1132         case ('F') :
1133         case ('E') : break;
1134         default :
1135             numofconflicts++;
1136         }
1137     }
1138     return (numofconflicts);
1139 }
1140
1141 void ClearChkLst(REQLSTNODE * lstptr)
1141 {
1142 /*
1143  * ARGUMENT
1144  *   (REQLSTNODE *)  lstptr - pointer to resource availability analysis list
1145  *
1146  * DESCRIPTION
1147  *   Clears list, leaves headnode
1148  *
1149  * RETURNS
1150  *
1151  * DATABASES AFFECTED
1152  *   none
1153  *
1154  */
1155
1156     REQLSTNODE * temp;
1157
1158     temp = lstptr;
1159     lstptr = lstptr->next;
1160     temp->next = (REQLSTNODE *) NULL;
1161     while (lstptr != (REQLSTNODE *) NULL)
1161     {
1162         temp = lstptr->next;
1163         lstptr->next = (REQLSTNODE *) NULL;
1164         free((REQLSTNODE *) lstptr);
1165         lstptr = temp;
1166     }
1167 }
1168
1169 void PrintChkLst(REQLSTNODE * lstptr)
1169 {
1170 /*
1171  * ARGUMENT
1172  *   (REQLSTNODE *)  lstptr - pointer to resource availability analysis list
1173  *
1174  * DESCRIPTION
1175  *   Prints list to stdout, used for debug
1176  *
1177  * RETURNS
1178  *
1179  * DATABASES AFFECTED
1180  *   none
1181  *
1182  */
1183
1184     printf("\n");
1185     while (lstptr != (REQLSTNODE *) NULL)
```

```
1185     {
1186         printf("\n  TYPE = >%c< recno = >%d< ", lstptr->type, lstptr->recno);
1187         printf(" AVAILSTATUS = >%c<", lstptr->availstatus);
1188         lstptr = lstptr->next;
1189     }
1190 }
1191
1192 int AllocMatls(REQLSTNODE * lstptr, long repid)
1192 {
1193 /*
1194  * ARGUMENT
1195  *   (REQLSTNODE *)  lstptr - pointer to resource availability analysis list
1196  *   (long)          repid  - repair id
1197  *
1198  * DESCRIPTION
1199  *   Allocates materials according to allocation condition information
1200  *   within the resource availability analysis list
1201  *
1202  * RETURNS
1203  *   (int)  status
1204  *
1205  */
1206     long matid, xmatid, xrepid, smatid;
1207     char smatdesc[30], sunit[5];
1208     float qntyneeded, xqty, sqty;
1209
1210     // loop until no more materials left to allocate
1211     while (lstptr->next != (REQLSTNODE *) NULL)
1211     {
1212         lstptr = lstptr->next;
1213         switch (lstptr->type)
1213         {
1214         case ('M') :
1215             /* PRIORITY OVERRIDE CASE */
1216             if ((lstptr->availstatus == 'O') && (lstptr->ovrd == TRUE))
1216             {
1217                 /* DUMP MATERIALS OF OVERRIDEN REPAIR BACK INTO SUPPLY */
1218                 if (d4select(matreq_dbf) < 0)
1218                 {
1219                     u4error(-1, "",(char *) 0);
1220                     return (RMRSERROR);
1221                 }
1222                 if (i4select(matreq_ndx1) < 0)
1222                 {
1223                     u4error(-2, "",(char *) 0);
1224                     return (RMRSERROR);
1225                 }
1226                 d4go(lstptr->resrecno);
1227                 xrepid = f4long(f4ref("REPID"));
1228                 d4top();
1229                 if (d4seek_double(xrepid))
1230                     u4error(-9999, "SEEK REPID RMRSERROR", "repid not found", "FN: AllocateMate
rials",(char *) 0);
1231                 // skip sequentially through mat reqs for subordinate repair
1232                 while ((f4long(f4ref("REPID")) == xrepid) && (! d4eof()))
1232                 {
1233                     xmatid = f4long(f4ref("MATID"));
1234                     xqty = f4double(f4ref("QTY"));
1235                     // search material supply on MATID
1236                     if (d4select(matsup_dbf) < 0)
1236                     {
1237                         u4error(-3, "",(char *) 0);
1238                         return (RMRSERROR);
1239                     }
1240                     i4select(-1);       //  to be replaced (CS$)
1241                     d4top();            //  to be replaced (CS$)
1242 /*                  i4select(matsup_ndx2);*/ /* to be indexed on MATID (CS$)*/
1243 /*                  d4seek_double(xmatid);*/ /* $$(CS$) */
1244                     while ((xmatid != f4long(f4ref("MATID"))) &&(! d4eof()))
1245                         d4skip(1);
1246                     f4r_double(f4ref("QTY"),(double) (f4double(f4ref("QTY")) + xqty));
1247                     d4write(d4recno());
1248                     if (d4select(matreq_dbf) < 0)
1248                     {
```

```
1249                        u4error(-5, "",(char *) 0);
1250                        return (RMRSERROR);
1251                    }
1252                    if (i4select(matreq_ndx1) < 0)
1252                    {
1253                        u4error(-6, "",(char *) 0);
1254                        return (RMRSERROR);
1255                    }
1256                    d4skip(1);
1257                }
1258                /* SET AVAIL STATUS TO 'O' (OVERRIDEN) */
1259                setrepairstatus(OVERRIDE, xrepid);
1260            }
1261            // MATERIAL SUBSTITUTION CASE
1262            else if ((lstptr->ovrd == FALSE) && (lstptr->availstatus != 'E'))
1262            {
1263                /* SUBSTITUTION OF MATERIAL for conflicting requirement */
1264                /* get mat info for substitute */
1265                d4select(matsup_dbf);
1266                d4go(lstptr->resrecno);
1267                sqty = lstptr->resqty;
1268                smatid = f4long(f4ref("MATID"));
1269                strcpy(smatdesc, f4str(f4ref("MATDESC")));
1270                strcpy(sunit, f4str(f4ref("UNIT")));
1271                /* replace matreq with the substitution material */
1272                d4select(matreq_dbf);
1273                d4go(lstptr->recno);
1274                f4r_long(f4ref("MATID"), smatid);
1275                f4r_str(f4ref("MATDESC"), smatdesc);
1276                f4r_double(f4ref("QTY"), sqty);
1277                f4r_str(f4ref("UNIT"), sunit);
1278                d4write(lstptr->recno);
1279            }
1280            // non-conflicting material requirements begin execution here
1281            /* ALLOCATE MATERIALS FROM SUPPLY AS USUAL */
1282            d4select(matreq_dbf);
1283            d4go(lstptr->recno);
1284            qntyneeded = f4double(f4ref("QTY"));
1285            matid = f4long(f4ref("MATID"));
1286            d4select(matsup_dbf);
1287            i4select(-1);               //  to be replaced (CS$)
1288            d4top();                    //  to be replaced (CS$)
1289 /*         i4select(matsup_ndx2);*//* MATID $$$(CS$)*/
1290 /*         d4seek_double(xmatid);*/ /* $$$ (CS$)*/
1291            while (matid != f4long(f4ref("MATID")))/* (CS$)$$$ MATID                */
1292                d4skip(1);
1293            f4r_double(f4ref("QTY"),(f4double(f4ref("QTY")) - qntyneeded));
1294            break;
1295        case ('E') : break;             // do nothing
1296        default : u4error(-7, "Invalid node TYPE"
1297                , "in resource requirement"
1298                , "linked list"
1299                , lstptr->type, (char *) 0);
1300        }
1301    }
1302    // set STATUS of repair id record to (P)ossible
1303    d4select(repair_dbf);
1304    d4seek_double(repid);
1305    f4r_char(f4ref("STATUS"), 'P');
1306    // set STATUS of material records to (A)llocated
1307    d4select(matreq_dbf);
1308    d4seek_double(repid);
1309    while (f4long(f4ref("REPID")) == repid)
1309    {
1310        f4r_char(f4ref("STATUS"), 'A');
1311        d4skip(1);
1312    }
1313    return (RMRSSUCCESS);
1314 }
1315
1316 int setrepairstatus(int tostatus, long repid)
1316 {
1317 /*
1318  * ARGUMENT
```

```
1319  *    (int) tostatus - predefined mneumonic constant corresponding to
1320  *                     the status to which the repair is changed to
1321  *                     (e.g. SUSPEND, XCANCEL, OVERRIDE)
1322  *    (long) repid   - repair id
1323  *
1324  * DESCRIPTION
1325  *   Changes STATUS fields of both the repair id record and associated
1326  *   resource requirement records to the appropriate code
1327  *
1328  * RETURNS
1329  *    (int) status code
1330  *
1331  * DATABASES AFFECTED (whether direct or indirect)
1332  *     repair_dbf (repair_ndx1 - REPID index       ) - d4seek used
1333  *     eqpreq_dbf (eqpreq_ndx1 - REPID index       ) - d4seek, d4skip used
1334  *     matreq_dbf (matreq_ndx1 - REPID index       ) - d4seek, d4skip used
1335  *
1336  */
1337
1338     float matqtytoreturn, supplyqty;
1339     long matid;
1340     char status;
1341
1342     switch (tostatus)
1342     {
1343     case (XCANCEL) : status = 'X';
1344         break;
1345     case (SUSPEND) : status = 'S';
1346         break;
1347     case (OVERRIDE) : status = 'O';
1348         break;
1349     default :;
1350     }
1351     d4select(repair_dbf);
1352     i4select(repair_ndx1);
1353     d4seek_double(repid);
1354     f4r_char(f4ref("STATUS"), status);
1355
1356     d4select(eqpreq_dbf);
1357     i4select(eqpreq_ndx1);
1358     d4seek_double(repid);
1359     while (f4long(f4ref("REPID")) == repid)
1359     {
1360         f4r_char(f4ref("STATUS"), status);
1361         d4skip(1);
1362     }
1363     d4select(matreq_dbf);
1364     i4select(matreq_ndx1);
1365     d4seek_double(repid);
1366     while (f4long(f4ref("REPID")) == repid)
1366     {
1367         if (f4char(f4ref("STATUS")) == 'A')
1367         {
1368             matqtytoreturn = f4double(f4ref("QTY"));
1369             matid = f4long(f4ref("MATID"));// $$ to be used w/ indexed search
1370             f4r_char(f4ref("STATUS"), status);
1371             d4select(matsup_dbf);
1372             i4select(-1);
1373             d4top();
1374             while ((matid != f4long(f4ref("MATID"))) &&(! d4eof()))
1375                 d4skip(1);              // $$ change to index search (ndx on "MATID")
1376             if (! d4eof())
1376             {
1377                 supplyqty = f4double(f4ref("QTY"));
1378                 f4r_double(f4ref("QTY"), supplyqty + matqtytoreturn);
1379             }
1380         }
1381         d4select(matreq_dbf);
1382         i4select(matreq_ndx1);
1383         f4r_char(f4ref("STATUS"), status);
1384         d4skip(1);
1385     }
1386     return (SUCCESS);
1387 }
```

```
1388
1389 //=============================================================================
1390 long GetFacnum(char * facnumstrptr)
1390 {
1391 /*
1392  * ARGUMENT
1393  *   (char *)  facnumstrptr - pointer to facility number string (e.g. "B4058")
1394  *
1395  * DESCRIPTION
1396  *   Accepts facility number string and returns a long representation
1397  *    ex. "B4058" -> (long)4058
1398  *
1399  * RETURNS
1400  *   facility number designation (long)
1401  *
1402  *
1403  */
1404     char tempstr[25];
1405     int i;
1406
1407     // strip 'B' off of facility designation and convert to int
1408     strcpy(tempstr, facnumstrptr);
1409     for (i = 1; tempstr[i] != '\0'; i++)
1410         tempstr[i - 1] = tempstr[i];
1411     tempstr[i - 1] = '\0';
1412
1413     return (atol(tempstr));
1414 }
1415
1416 long GetNxtAvlRepid(void)
1416 {
1417 /*
1418  * ARGUMENT: none
1419  *
1420  * DESCRIPTION
1421  *   Gets the next available RMRS repair id number; checks for
1422  *   empty database situation
1423  *
1424  * RETURNS
1425  *   (long) repair id number
1426  *
1427  * DATABASES AFFECTED (whether direct or indirect)
1428  *    repair_dbf (no index used)
1429  *
1430  */
1431
1432     // assign unique RMRS id # to repair
1433     d4select(repair_dbf);
1434     i4select(-1);
1435     if (d4bottom() == 3)
1436         return (1);
1437     else
1438         return (f4long(f4ref("REPID")) + 1);
1439 }
1440
1441 int AddRepRec(long facnum, long repid, int priority)
1441 {
1442 /*
1443  * ARGUMENT
1444  *   (long)      facnum    - facility number
1445  *   (long)      repid     - repair id
1446  *   (int)       priority  - AB mission-critical facility priority assignment
1447  *
1448  * DESCRIPTION
1449  *   Adds a repair data record to the end of REPAIR.DBF
1450  *
1451  * RETURNS
1452  *   (int) status of d4append() operation
1453  *
1454  * DATABASES AFFECTED (whether direct or indirect)
1455  *    repair_dbf (no index used)
1456  *
1457  */
1458
```

```
1459     // add repair record to REPAIR.DBF
1460     d4select(repair_dbf);
1461     f4r_long(f4ref("FACNUM"), facnum);
1462     f4r_long(f4ref("REPID"), repid);
1463     f4r_int(f4ref("PRIORITY"), priority);
1464     f4r_int(f4ref("SCHDPRTY"), priority);
1465     f4r_char(f4ref("STATUS"), 'N');
1466
1467     return (d4append());
1468 }
1469
1470 void cleanwin(void)
1470 {
1471     int i;
1472     for (i = 1; i <= 7; i++)
1473         vatputf(win, i, 0, "                                                    ");
1474 }
1475
1476
1477 int GetRepairPrty(long repid)
1477 {
1478 /*
1479  * ARGUMENT
1480  *   (long) repid - repair id
1481  *
1482  * DESCRIPTION
1483  *   Gets priority of repair
1484  *
1485  * RETURNS
1486  *   (int) priority
1487  *
1488  * DATABASES AFFECTED (whether direct or indirect)
1489  *    repair_dbf (repair_ndx1 - REPID index       ) - d4seek used
1490  *
1491  */
1492
1493     int prty;
1494
1495     d4select(repair_dbf);
1496     i4select(repair_ndx1);
1497     d4seek_double(repid);
1498     prty = f4int(f4ref("PRIORITY"));
1499     return (prty);
1500 }
```

```
 1 /*
 2  *  Procutil.c
 3  *
 4  *     functions used by the PROCESS menu selections
 5  *
 6  */
 7
 8
 9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <dos.h>
13
14 #include "sysdefs.h"
15 #include "d4base.h"
16 #include "u4error.h"
17 #include "miscutil.h"
18
19 #include "dw.h"
20
21 HWND xdsplymessbox(void);
22
23 int move_file(char * oldname, char * newname)
23 {
24 /*
25  * ARGUMENT
26  *    (char *) oldname - old filename string
27  *    (char *) newname - new filename string
28  *
29  * DESCRIPTION
30  *    Changes the filename specification by modifying the FCB
31  *
32  * RETURNS
33  *    (int) status code:
34  *
35  *          interrupt 56H status codes : 3 - Path not found
36  *                                       5 - Access denied
37  *                                      17 - Path not found
38  *
39  */
40
41     union REGS regs; struct SREGS sregs;
42     int ret;
43     regs.h.ah = 0x56;
44     sregs.ds = FP_SEG(oldname);
45     regs.x.dx = FP_OFF(oldname);
46     sregs.es = FP_SEG(newname);
47     regs.x.di = FP_OFF(newname);
48     ret = intdosx(& regs, & regs, & sregs);
49
50     /* if carry flag is set, there was an error */
51     ret = regs.x.ax;
52     return (regs.x.cflag ? ret : 0);
53 }
54
55 int iseqpconf(REQLSTNODE * lstptr)
55 {
56 /*
57  * ARGUMENT
58  *    (REQLSTNODE *)  lstptr - pointer to resource availability analysis list
59  *
60  * DESCRIPTION
61  *    Checks the resource availability analysis list to see if any equipment
62  *    conflicts exist
63  *
64  * RETURNS
65  *    (int) TRUE or FALSE
66  *
67  */
68     if (! lstptr)
68     {
69         u4error(EMPTYLST, "Unexpected empty list error", (char *) 0);
70         return (FALSE);                /* error                                */
71     }
```

```
72      for (lstptr = lstptr->next; lstptr->type == 'E'; lstptr = lstptr->next)
73          if (lstptr->availstatus != 'F')
74              return (TRUE);              /* eqp conf found                    */
75      return (FALSE);                     /* no conflicts                      */
76  }
77
78  HWND xdsplymessbox(void)
78  {
79  /*
80   * ARGUMENT
81   *
82   * DESCRIPTION
83   *   Sets up and displays a standard message box
84   *
85   * RETURNS
86   *   (HWND) Handle to window of message box
87   *
88   */
89
90      HWND messwin;
91
92      messwin = vcreat(8, 30, GREYBOX, YES);
93      vlocate(messwin, 8, 24);
94      vframe(messwin, GREYBOX, FRDOUBLE);
95      vshadow(messwin, CURRENT, SHADOW100, BOTTOMRIGHT);
96
97      visible(messwin, YES, YES);
98      return (messwin);
99  }
100
101
102 int askcomp(void)
102 {
103 /*
104  * ARGUMENT
105  *
106  * DESCRIPTION
107  *   Asks the user if a COMPROMISE should be attempted
108  *
109  * RETURNS
110  *   (int) TRUE or FALSE
111  *
112  */
113
114     HWND win;
115     unsigned int key;
116
117     win = xdsplymessbox();
118     vratps(1, 0, GRYBTXT, "  //  REPAIR HAS MATERIAL  .. ");
119     modattr(win, 1, 2, 2, SELQTYBLINK);
120     modattr(win, 1, 27, 2, SELQTYBLINK);
121     vratps(2, 0, GRYBTXT, "           CONFLICTS          ");
122     vratps(4, 0, GRYBTXT, " substitution may be possible ");
123     vratps(6, 0, KEYMESS, "    Do you wish to enter      ");
124     vratps(7, 0, KEYMESS, "    COMPROMISE mode (N):  ?   ");
125     modattr(win, 7, 25, 3, FLSHCHR);
126     while (1)
126     {
127         key = getkey();
128         switch (key)
128         {
129         case ('N') :
130         case ('n') :
131         case (ENTR) : vratps(7, 26, HELP, "N");
132             delay(2);
133             vdelete(win, NONE);
134             dmpxtrakeys();
135             return (FALSE);
136         case ('Y') :
137         case ('y') : vratps(7, 26, HELP, "Y");
138             delay(2);
139             vdelete(win, NONE);
140             dmpxtrakeys();
141             return (TRUE);
```

```
142         default : vbeep();              /* INVALID keystroke                */
143         }
144     }
145 }
146
147
148
```

```
 1 /*******************************************************
 2  *                                                     *
 3  *  reset.c                                            *
 4  *                                                     *
 5  *    This program will ZAP all the records in the     *
 6  *    databases that are populated by processed        *
 7  *    repairs and reset all material supply            *
 8  *    quantities to 1000.0.                            *
 9  *                                                     *
10  *   by Chris Sawyer 10/6/91                           *
11  *******************************************************/
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include "d4base.h"
16 #include "rmrsutil.h"
17
18 int reset(void)
19 {
20
21     open_repair(0);
22     d4zap(1L, d4reccount());
23     d4close_all();
24
25     open_matreq(0);
26     d4zap(1L, d4reccount());
27     d4close_all();
28
29     open_eqpreq(0);
30     d4zap(1L, d4reccount());
31     d4close_all();
32
33     open_repinfo(0);
34     d4zap(1L, d4reccount());
35     d4close_all();
36
37     open_matsup(0);
38     d4top();
39     while (! d4eof())
39     {
40         f4r_double(f4ref("QTY"), 1000.0);
41         d4skip(1);
42     }
43     d4pack();
44     d4close_all();
45
46     open_eqpsup(0);
47     d4pack();
48     d4close_all();
49     fcloseall();
50 }
```

```
#include "d4all.h"
#include "w4.h"
#include "g4char.h"
#include "p4misc.h"
#include "rmrs.h"

#include <string.h>
#include <ctype.h>

#include "rmrsclr.h"

static void b4set_row(int, long);
static int  b4modify(int);
static int  b4skip_one(int);
static int  b4page(int);
static int  b4top(int);
static int  b4find(int);
static int  b4bottom(int);
static int  b4char(void);
static void b4edit_setup(void);
static void b4browse_setup(void);
static int  b4setup(int (*) (void), int (*) (void));
static void b4deact_menu(void);
static int  b4add_copy(int);
static int  b4select_index(int);
static int  b4delete(int);
static int  b4undelete(int);
static int  b4go_rec(void);             /* Goes to the record at 'b4cursor_row'.        */
static int  b4empty_check(void);

static int (* user_edit_setup) (void) = 0;
static int (* user_browse_setup) (void) = 0;
static int  b4on_browse = -1;
static int  b4cur_ref = -1;

static int  b4pull_ref = -1;
static int  b4select_ref = -1;

static int  b4page_size = -1;
static int  b4top_margin = 3;
static int  b4cur_top_margin = -1;
static int  b4bottom_margin = 1;
static int  b4cursor_col = -1;
static int  last_read_char = 0;
static int  is_empty = 0;
static int  do_escape = 0;

/* Action routines assume that the record at 'b4cursor_row' is
   'b4cursor_rec'. */
static int  b4cursor_row;
static long b4cursor_rec;               /* The record at 'b4cursor_row'.  This info
                                           is filled in by 'b4display_recs'.
                                           It becomes '-1L' if there was no record
                                           at the cursor.                           */
static long b4start_rec;                /* Saved by 'b4display_assume'.
                                           This is the starting record number.      */
static int  b4num_displayed;            /* Saved by 'b4display_recs'.               */

static int  b4display_one(int);
static int  b4display_assume(void);     /* Assumes that 'b4cursor_rec'
                                           is at 'b4cursor_row'.                    */
static int  b4display_recs(long);
static void b4display_cursor(void);

extern CB_WINDOW * v4window_ptr;
extern GET * v4get;
extern INDEX * v4index;


static long natt1 = KW, natt2 = WK, matt1 = KW, matt2 = WK;


static int  b4go_rec()
{
```

```
    if (! is_empty)
        if (b4cursor_rec != d4recno())
            d4go(b4cursor_rec);
    return 0;
}

static int  b4empty_check()
{
    if (d4eof())
        d4top();
    else
    {
        if (d4skip(1L) == -3)
            d4top();
        else
            d4skip(-1L);
    }
    is_empty = 1;
    b4cursor_rec = 0L;
    if (! d4eof())
    {
        is_empty = 0;
        b4cursor_rec = d4recno();
    }
    return 0;
}

static int (* verify_routine) (int) = 0;
#ifdef KR
   void  b4verify( verify_rou )
   int (*verify_rou)() ;
#else
    void  b4verify(int (* verify_rou) (int))
#endif
{
    verify_routine = verify_rou;
}

static int (* call_routine) (int, int) = 0;
#ifdef KR
   void  b4call( call_rou )
   int (*call_rou)() ;
#else
    void  b4call(int (* call_rou) (int, int))
#endif
{
    call_routine = call_rou;
}


static int  temp_ref = -1;

static void  b4deact_menu()
{
    if (w4select(-1) != b4pull_ref && w4select(-1) != b4cur_ref)
    {
        temp_ref = w4select(-1);
        n4refresh(temp_ref);
        w4deactivate(temp_ref);
    }
}

static void  b4set_row(int new_row, long new_attribute)
{
    int get_on;
    GET * get_ptr;

    new_row += b4cur_top_margin;

    for (get_on = v4window_ptr->first_get; get_on >= 0; get_on = get_ptr->next)
    {
        get_ptr = v4get + get_on;
        if (b4on_browse)
            get_ptr->row = new_row;
```

```
149         get_ptr->attribute = new_attribute;
150     }
151 }
152
153 static int  b4modify(int is_modify)
154 {
155     int  rc;
156
157     if (is_modify)
158         if (is_empty) return 0;
159
160     b4deact_menu();
161     if (is_modify)
162         b4go_rec();
163
164     w4activate(b4cur_ref);
165     b4set_row(b4cursor_row, KW);
166
167     for (;;)
168     {
169         if (is_modify)
170             d4unlock(-1L);
171
172         last_read_char = g4read();
173         if (last_read_char == ESC || last_read_char == CTRL_Q)
174         {
175             last_read_char = 0;
176             if (is_modify)
177                 d4go(b4cursor_rec);
178             else
179                 return ESC;
180
181             break;
182         }
183         else
184         {
185             if (! u4ptr_equal((void *) 0, (void *) verify_routine))
186                 rc = (* verify_routine) (b4cur_ref);
187             else
188                 rc = 0;
189
190             if (rc == 0)
191             {
192                 if (is_modify) d4lock(d4recno(), 1);
193                 if (! u4ptr_equal((void *) call_routine, (void *) 0))
194                     (* call_routine) (b4cur_ref, b4cursor_row + b4cur_top_margin);
195
196                 if (is_modify)
197                     rc = d4write(d4recno());
198                 else
199                     rc = d4append();
200                 if (rc == -3) continue;
201
202                 d4flush(d4select(-1));
203                 if (last_read_char == RETURN || last_read_char == DOWN)
204                     last_read_char = 0;
205                 break;
206             }
207         }
208     }
209
210     if (is_modify)
211     {
212         b4display_assume();
213         b4display_cursor();
214     }
215
216     d4unlock(-1L);
217     return 0;
218 }
219
220 static int b4add_copy(int is_blank)
221 {
222     int  rc, save_row;
```

```
223
224     if (is_blank)
225         d4go(0L);
226     else
227         b4go_rec();
228
229     save_row = b4cursor_row;
230     b4cursor_row = b4num_displayed;
231     if (b4cursor_row >= b4page_size)
232         b4cursor_row--;
233
234     rc = b4modify(0);
235     if (rc != 0)
236         b4cursor_row = save_row;
237     else
238     {
239         b4cursor_rec = d4recno();
240         b4go_rec();
241         b4empty_check();
242     }
243
244     b4display_assume();
245     return 0;
246 }
247
248
249 static void b4display_cursor()
250 {
251     w4select(b4cur_ref);
252     w4cursor(b4cur_top_margin + b4cursor_row, b4cursor_col);
253 }
254
255 static int  b4display_one(int disp_row)
256 {
257     int  first_get;
258
259     b4set_row(disp_row, KC);
260     first_get = v4window_ptr->first_get;
261
262     if (! u4ptr_equal((void *) call_routine, (void *) 0))
263         (* call_routine) (b4cur_ref, disp_row + b4cur_top_margin);
264
265     if (d4deleted())
266         w4(v4get[first_get].row, v4get[first_get].col - 1, "*");
267     else
268         w4(v4get[first_get].row, v4get[first_get].col - 1, " ");
269
270     g4display();
271     return 0;
272 }
273
274 /* Returns the number of records displayed.
275    Stores record at cursor in 'b4cursor_rec'.
276 */
277
278 static int  b4display_recs(long start_rec)
279 {
280     int  i;
281
282     b4num_displayed = 0;
283     w4activate(b4cur_ref);
284
285     b4cursor_rec = -1L;
286     if (is_empty)
287         d4go(0L);
288     else
289         d4go(start_rec);
290
291     for (i = 0; i < b4page_size; i++)
292     {
293         b4display_one(i);
294
295         if (! d4eof())
296         {
```

```
297             b4num_displayed++;
298             if (i == b4cursor_row)
299                 b4cursor_rec = d4recno();
300
301             d4skip(1L);
302         }
303     }
304
305     return 0;
306 }
307
308 static int  b4display_assume()
309 {
310     b4start_rec = d4recno();
311     b4go_rec();
312     d4skip((long) - b4cursor_row);
313     b4display_recs(d4recno());
314     b4display_cursor();
315
316     return 0;
317 }
318
319 static int  b4top(int junk_parm)
320 {
321     d4top();
322     b4cursor_row = 0;
323     b4cursor_rec = d4recno();
324
325     b4deact_menu();
326     b4display_assume();
327
328     d4unlock(-1L);
329     return 0;
330 }
331
332 static int  b4bottom(int junk_parm)
333 {
334     long  bot_rec;
335
336     d4bottom();
337     bot_rec = d4recno();
338     d4skip((long) (1 - b4page_size));
339
340     b4deact_menu();
341     b4display_recs(d4recno());
342     b4cursor_rec = bot_rec;
343     b4cursor_row = b4num_displayed - 1;
344     b4display_cursor();
345
346     d4unlock(-1L);
347     return 0;
348 }
349
350 static int  b4skip_one(int n)
351 {
352     int  rc;
353
354     b4go_rec();
355     rc = d4skip((long) n);
356     if (rc < 0) return - 1;
357     if (rc == 1) return 0;
358     if (rc == 3) return 0;
359
360     b4cursor_row += n;
361     b4cursor_rec = d4recno();
362     if (b4cursor_row < 0)
363     {
364         b4cursor_row = 0;
365         b4deact_menu();
366         b4display_assume();
367         return 0;
368     }
369
370     if (b4cursor_row >= b4page_size)
```

```
371     {
372         b4cursor_row = b4page_size - 1;
373         b4deact_menu();
374         b4display_assume();
375         return 0;
376     }
377
378     b4display_cursor();
379     d4unlock(-1L);
380     return 0;
381 }
382
383 static int  b4page(int no_recs)
384 {
385     b4go_rec();
386     if (no_recs < 0 && b4cursor_row > 0)
387     {
388         d4skip((long) - b4cursor_row);
389         b4cursor_rec = d4recno();
390         b4cursor_row = 0;
391         b4display_cursor();
392         return 0;
393     }
394
395     if (no_recs > 0 && b4cursor_row < b4num_displayed - 1)
396     {
397         d4skip((long) (b4num_displayed - 1 - b4cursor_row));
398         b4cursor_rec = d4recno();
399         b4cursor_row = b4num_displayed - 1;
400         b4display_cursor();
401         return 0;
402     }
403
404     d4skip((long) no_recs);
405     if (d4eof()) d4skip(-1L);
406     b4cursor_rec = d4recno();
407     b4deact_menu();
408     b4display_assume();
409     d4unlock(-1L);
410     return 0;
411 }
412
413 void  b4margin(int top_margin, int bottom_margin)
414 {
415     b4top_margin = top_margin;
416     b4bottom_margin = bottom_margin;
417 }
418
419 static int  b4help(int junk_parm)
420 {
421     int w_ref, startrow = 1, startcol = 1;
422
423     w_ref = w4define(2, 5, 23, 65);
424     w4popup();
425     w4attribute(natt1);
426     w4border(DOUBLE, natt1);
427     w4title(0, 2, " Help ", natt1);
428     w4activate(w_ref);
429
430     w4(startrow++, startcol, "You can use the menu to select an option or you can press");
431     w4(startrow++, startcol, "one of the following command keys:");
432     w4(startrow++, startcol, "  ");
433     w4(startrow++, startcol, "A  Add a blank record.");
434     w4(startrow++, startcol, "B  Move to the bottom database record.");
435     w4(startrow++, startcol, "C  Add a record by copying the current record.");
436     w4(startrow++, startcol, "D  Mark the current record for deletion.");
437     w4(startrow++, startcol, "E  Switch to the edit screen if possible.");
438     w4(startrow++, startcol, "F  Find a record.");
439     w4(startrow++, startcol, "H  Display this help screen.");
440     w4(startrow++, startcol, "M  Modify the current record.");
441     w4(startrow++, startcol, "R  Enter a record to move to.");
442     w4(startrow++, startcol, "S  Select a record ordering index.");
443     w4(startrow++, startcol, "T  Move to the top database record.");
444     w4(startrow++, startcol, "U  Remove the deletion mark from the current record.");
```

```
445     w4(startrow++, startcol, "Z  Switch to the browse screen if possible.");
446     w4(startrow++, startcol, "X  Exit.");
447     w4(startrow++, startcol, "<PgUp>  or <PgDn>   Move up/down one screen of records.");
448     w4(startrow++, startcol, "<Up or Down Arrow>  Move to the previous/next record.");
449     getch();
450     w4deactivate(w_ref);
451     w4close(w_ref);
452     return  0;
453 }
454
455 static int  b4record(int junk_parm)
456 {
457     long  rec;
458     int   rc, w_ref;
459
460     rec = b4cursor_rec;
461     if (rec < 0L) rec = 0L;
462
463     w_ref = w4define(9, 20, 14, 58);
464     w4attribute((long) KW);
465     w4border(DOUBLE, (long) KW);
466     w4popup();
467     w4title(0, -1, " Record Number Command ", B_WHITE);
468     w4activate(w_ref);
469
470     w4(1, 2, "Enter Record Number: ");
471     g4attribute((long) WK);
472     g4long(w4row(), w4col(), & rec);
473     w4(2, 2, "Records in Database:");
474     w4long(w4row(), w4col(), d4reccount(), 9);
475     rc = g4read();                      /* Read the Record                    */
476
477     w4deactivate(w_ref);
478     w4close(w_ref);
479
480     if (rc != ESC)
481     {
482         if (rec > 0 && rec <= d4reccount())
483         {
484             if (is_empty)
485             {
486                 w4display(" Message ",
487                     "To display a record, there will be an immediate",
488                     "switch to record number ordering.", (char *) 0);
489                 i4unselect();
490                 b4empty_check();
491             }
492             b4cursor_rec = rec;
493             b4cursor_row = 0;
494
495             b4deact_menu();
496             b4display_assume();
497         }
498     }
499
500     b4display_cursor();
501     d4unlock(-1L);
502     return 0;
503 }
504
505 static int  b4find(int junk_parm)
506 {
507     int  index_ref, rc, w_ref;
508     char seek_data[101];
509
510     index_ref = i4seek_ref();
511     if (index_ref < 0) return 0;
512
513     w_ref = w4define(9, 21, 16, 58);
514     w4attribute((long) KW);
515     w4popup();
516     w4border(DOUBLE, (long) KW);
517     w4title(0, -1, " Find ",(long) KW);
518     w4activate(w_ref);
```

```
519
520     w4(1, 3, "Find Index: ");
521     w4(1, w4col(), i4name(index_ref));
522
523     memset(seek_data, 0, (size_t) sizeof (seek_data));
524     w4(3, 3, "Enter the Find Information:");
525     g4attribute((long) WK);
526     g4(4, 3, seek_data);
527     if (i4type(index_ref) == 'N')
528         g4width(24, 24);
529     else
530         g4width(24, 100);
531     rc = g4read();
532
533     w4deactivate(w_ref);
534     w4close(w_ref);
535
536     if (rc == ESC)
537         b4display_cursor();
538     else
539     {
540         if (i4type(index_ref) == 'N')
541         {
542             if ((rc = d4seek_double(c4atod(seek_data, 24))) < 0) return - 1;
543         }
544         else
545             if ((rc = d4seek_str(seek_data)) < 0) return - 1;
546
547         if (rc == 3)
548         {
549             c4trim_n(seek_data, (int) sizeof (seek_data));
550             w_ref = w4define(6, 5, 10,(46 + strlen(seek_data) + 8));
551             w4popup();
552             w4border(DOUBLE, natt1);
553             w4attribute(natt1);
554             w4activate(w_ref);
555             w4(1, 1, "The end of the file was reached finding data: ");
556             w4(1, w4col(), seek_data);
557             g4char();
558             w4deactivate(w_ref);
559             w4close(w_ref);
560             b4display_cursor();
561
562             d4unlock(-1L);
563             return 0;
564         }
565
566         b4cursor_rec = d4recno();
567         b4cursor_row = 0;
568         b4deact_menu();
569         b4display_assume();
570     }
571
572     d4unlock(-1L);
573     return 0;
574 }
575
576
577 static int  b4select_index(int item_ref)
578 {
579     i4select(n4int_get(item_ref));
580     n4start_item(item_ref);
581     last_read_char = -2;
582     do_escape = 1;
583     return 0;
584 }
585
586 static int  b4delete(int junk_parm)
587 {
588     if (b4cursor_rec > 0L)
589         d4delete(b4cursor_rec);
590
591     b4deact_menu();
592     w4select(b4cur_ref);
```

```
593     b4display_one(b4cursor_row);
594     d4unlock(-1L);
595     return 0;
596 }
597
598 static int  b4undelete(int junk_parm)
599 {
600     if (b4cursor_rec > 0L)
601         d4recall(b4cursor_rec);
602
603     b4deact_menu();
604     w4select(b4cur_ref);
605     b4display_one(b4cursor_row);
606     d4unlock(-1L);
607     return 0;
608 }
609
610 static int  b4browse_flip(int junk_parm)
611 {
612     if (u4ptr_equal((void *) user_edit_setup, (void *) 0)) return 0;
613     if (u4ptr_equal((void *) user_browse_setup, (void *) 0)) return 0;
614
615     b4deact_menu();
616     n4refresh(b4pull_ref);
617
618     w4close(b4cur_ref);
619
620     if (b4on_browse)
621         b4edit_setup();
622     else
623         b4browse_setup();
624
625     return 0;
626 }
627
628 #ifdef KR
629    int  b4edit( browse_setup, edit_setup )
630    int (*browse_setup)() ;
631    int (*edit_setup)() ;
632 #else
633     int  b4edit(int (* browse_setup) (void), int (* edit_setup) (void))
634 #endif
635 {
636     b4on_browse = 0;
637     return (b4setup(browse_setup, edit_setup));
638 }
639
640 #ifdef KR
641    int  b4browse( browse_setup, edit_setup )
642    int (*browse_setup)() ;
643    int (*edit_setup)() ;
644 #else
645     int  b4browse(int (* browse_setup) (void), int (* edit_setup) (void))
646 #endif
647 {
648     b4on_browse = 1;
649     return (b4setup(browse_setup, edit_setup));
650 }
651
652 static char browse_or_edit[] = "BROWSE";
653
654 static void b4edit_setup()
655 {
656     b4cur_ref = (* user_edit_setup) ();
657     b4on_browse = 0;
658     b4cur_top_margin = v4get[v4window_ptr->first_get].row;
659     b4page_size = 1;
660     b4cursor_row = 0;
661     b4cursor_col = v4get[v4window_ptr->first_get].col;
662     strcpy(browse_or_edit, "Browse");
663     b4display_assume();
664     d4unlock(-1L);
665 }
666
```

```
667 static void  b4browse_setup()
668 {
669     b4cur_ref = (* user_browse_setup) ();
670     b4on_browse = 1;
671     b4cur_top_margin = b4top_margin;
672     b4page_size = w4height(-1) - b4top_margin - b4bottom_margin;
673     b4cursor_row = 0;
674     b4cursor_col = v4get[v4window_ptr->first_get].col;
675     strcpy(browse_or_edit, "Edit  ");
676     b4display_assume();
677     d4unlock(-1L);
678 }
679
680 #ifdef KR
681         static int  b4setup( browse_setup, edit_setup )
682         int (*browse_setup)() ;
683         int (*edit_setup)() ;
684 #else
685     static int  b4setup(int (* browse_setup) (void), int (* edit_setup) (void))
686 #endif
687 {
688     int  srch_ref, pos_ref, add_ref, ch_ref, fillwin;
689     int  i_ref, item_ref;
690     int  start_lock_code;
691
692 /*   w4clear(-1) ;*/
693
694     user_edit_setup = edit_setup;
695     user_browse_setup = browse_setup;
696     start_lock_code = d4lock_code(2);
697
698     b4empty_check();
699
700     if (b4on_browse)
701         b4browse_setup();
702     else
703         b4edit_setup();
704
705     b4select_ref = -1;
706
707     /* Define the index file selection menu. */
708     if (d4ptr()->index_ref >= 0)
709     {
710         i_ref = h4first((char **) & v4index, d4ptr()->index_ref);
711
712         b4select_ref = w4define(-1, -1, -1, -1);
713         w4attribute(natt1);
714         w4border(SINGLE, natt1);
715         n4attribute(natt1, natt2);
716         n4("Record Number Ordering");
717         n4action(b4select_index);
718         n4int_save(-1);
719         for (; i_ref >= 0; i_ref = v4index[i_ref].next)
720         {
721             item_ref = n4(v4index[i_ref].name);
722             if (i4seek_ref() == i_ref)
723                 n4start_item(item_ref);
724             n4action(b4select_index);
725             n4int_save(i_ref);
726         }
727     }
728
729     /* Define the main pulldown menu. */
730     n4key_special(-1, CTRL_C, -1, -1);
731     b4pull_ref = w4define(0, 0, 0, 79);
732     w4attribute(matt1);
733     n4attribute(matt1, matt2);
734     n4key_special(ESC, CTRL_C, -1, -1);
735
736     n4(" Help ");
737     n4key(0, 0, -1);
738     n4action(b4help);
739
740     n4(" Modify ");
```

```
741     n4key(0, 0, -1);
742     n4action(n4sub_menu);
743     n4ptr_save(& ch_ref);
744
745     n4(" Add ");
746     n4key(0, 0, -1);
747     n4action(n4sub_menu);
748     n4ptr_save(& add_ref);
749
750     n4(" Position ");
751     n4key(0, 0, -1);
752     n4action(n4sub_menu);
753     n4ptr_save(& pos_ref);
754
755     if (b4select_ref >= 0)
756     {
757         n4(" Find ");
758         n4key(0, 0, -1);
759         n4action(n4sub_menu);
760         n4ptr_save(& srch_ref);
761     }
762
763     if (! u4ptr_equal((void *) user_edit_setup, (void *) 0) &&
764             ! u4ptr_equal((void *) user_edit_setup, (void *) 0))
765     {
766         n4(browse_or_edit);
767         n4key(0, 0, -1);
768         n4action(b4browse_flip);
769     }
770
771     n4(" Exit ");
772     n4key(0, 0, -1);
773     n4parm(-1);
774
775     ch_ref = w4define(-1, -1, -1, -1);
776     w4attribute(natt1);
777     w4border(SINGLE, natt1);
778     n4attribute(natt1, natt2);
779     n4(" Modify Record "); n4action(b4modify); n4parm(1);
780     n4(" Delete Record "); n4action(b4delete);
781     n4(" Undelete Record "); n4action(b4undelete);
782
783     if (b4select_ref >= 0)
784     {
785         srch_ref = w4define(-1, -1, -1, -1);
786         w4attribute(natt1);
787         w4border(SINGLE, natt1);
788         n4attribute(natt1, natt2);
789         n4(" Find "); n4action(b4find);
790         if (b4select_ref >= 0)
791         {
792             n4(" Select Record Ordering Index ");
793             n4action(n4sub_menu);
794             n4ptr_save(& b4select_ref);
795         }
796     }
797
798     pos_ref = w4define(-1, -1, -1, -1);
799     w4attribute(natt1);
800     w4border(SINGLE, natt1);
801     n4attribute(natt1, natt2);
802     n4(" Record "); n4action(b4record);
803     n4(" Top "); n4action(b4top);
804     n4(" Bottom "); n4action(b4bottom);
805
806     add_ref = w4define(-1, -1, -1, -1);
807     w4attribute(natt1);
808     w4border(SINGLE, natt1);
809     n4attribute(natt1, natt2);
810     n4(" Add Blank "); n4action(b4add_copy); n4parm(1);
811     n4(" Add Copy "); n4action(b4add_copy); n4parm(0);
812     n4key((int) 'C', 1, 4);
813
814     n4pulldown(b4pull_ref);
```

```
815     w4select(b4pull_ref);
816     w4memory();
817     fillwin = w4define(0, 0, 0, 79);
818     w4attribute(matt1);
819     w4activate(fillwin);
820
821     n4char_routine(b4char);
822     n4activate(b4pull_ref);
823
824     w4close(b4cur_ref);
825
826     w4deactivate(b4pull_ref);
827     w4close(b4pull_ref);
828     w4close(fillwin);
829     n4char_routine(0);
830
831     if (b4select_ref >= 0)
832     {
833         w4close(b4select_ref);
834         w4close(srch_ref);
835     }
836
837     w4close(pos_ref);
838     w4close(add_ref);
839     w4close(ch_ref);
840
841     d4lock_code(start_lock_code);
842     w4clear(-1);
843
844     return 0;
845 }
846
847 static int  b4char()
848 {
849     int  rc;
850
851     if (do_escape)
852     {
853         do_escape = 0;
854         return ESC;
855     }
856
857     if (last_read_char != 0)
858     {
859         rc = last_read_char;
860         last_read_char = 0;
861     }
862     else
863         rc = g4char();
864
865     if (rc > 0 && rc < 0xFF)
866         rc = u4toupper(rc);
867
868     if (w4select(-1) != b4pull_ref)
869         if (rc >= (int) 'A' && rc <= (int) 'Z') return rc;
870
871     switch (rc)
872     {
873     case PGUP :
874         b4page(- b4page_size);
875         break;
876
877     case PGDN :
878         b4page(b4page_size);
879         break;
880
881         #ifndef UNIX
882         case ALT_A :
883         #endif
884     case 'A' :
885         b4add_copy(1);
886         break;
887
888         #ifndef UNIX
```

```
889         case ALT_B :
890         #endif
891     case 'B' :
892         b4bottom(0);
893         break;
894
895         #ifndef UNIX
896         case ALT_C :
897         #endif
898     case 'C' :
899         b4add_copy(0);
900         break;
901
902         #ifndef UNIX
903         case ALT_D :
904         #endif
905     case 'D' :
906         b4delete(0);
907         break;
908
909         #ifndef UNIX
910         case ALT_E :
911         #endif
912     case 'E' :
913         if (b4on_browse)
914             b4browse_flip(0);
915         break;
916
917         #ifndef UNIX
918         case ALT_F :
919         #endif
920     case 'F' :
921         b4find(0);
922         break;
923
924         #ifndef UNIX
925         case ALT_H :
926         #endif
927     case 'H' :
928         b4help(0);
929         break;
930
931         #ifndef UNIX
932         case ALT_M :
933         #endif
934     case 'M' :
935         b4modify(1);
936         break;
937
938         #ifndef UNIX
939         case ALT_R :
940         #endif
941     case 'R' :
942         b4record(0);
943         break;
944
945         #ifndef UNIX
946         case ALT_S :
947         #endif
948     case 'S' :
949         if (b4select_ref >= 0)
950             n4activate(b4select_ref);
951         break;
952
953     case - 2 :                          /* After 'b4select_index' to refresh the display.*/
954         b4deact_menu();
955         b4go_rec();
956         b4empty_check();
957         b4display_assume();
958         if (b4cursor_rec == -1L)
959         {
960             b4cursor_rec = b4start_rec;
961             b4cursor_row = 0;
962         }
```

```
963         d4unlock(-1L);
964         break;
965
966         #ifndef UNIX
967         case ALT_T :
968         #endif
969     case 'T' :
970         b4top(0);
971         break;
972
973         #ifndef UNIX
974         case ALT_U :
975         #endif
976     case 'U' :
977         b4undelete(0);
978         break;
979
980         #ifndef UNIX
981         case ALT_Z :
982         #endif
983     case 'Z' :
984         if (! b4on_browse)
985             b4browse_flip(0);
986         break;
987
988         #ifndef UNIX
989         case ALT_X :
990         #endif
991     case 'X' :
992         return CTRL_C;
993
994     case UP :
995         if (v4window_ptr->horizontal)
996             b4skip_one(-1);
997         return rc;
998
999     case DOWN :
1000        if (v4window_ptr->horizontal)
1001            b4skip_one(1);
1002        return rc;
1003
1004    default :
1005        return rc;
1006    }
1007
1008    return 0;
1009 }
1010
1011 int  b4quick_browse()
1012 {
1013    int  c, j, i, next_c, w, fillwin;
1014    long ref;
1015    int ur = 1, lc = 0, lr = 24, rc = 79;
1016    char vl[2] =
1016     {
1016         179, 0
1016     };
1017    char vt[2] =
1017     {
1017         194, 0
1017     };
1018
1019
1020    w4define(ur, lc, lr, rc);
1021    w4attribute(KC);
1022    w4memory();
1023
1024    w4activate(-1);
1025    g4release(0);
1026
1027    next_c = 2;
1028
1029    w4repeat(2, lc + 2, 196, (rc - lc - 3));
1030    w4(0, 2, d4name());
```

```
1031     for (j = 1; j <= f4num_fields(); j++)
1032     {
1033         c = next_c;
1034
1035
1036         ref = f4j_ref(j);
1037         if (f4type(ref) == 'M') continue;
1038         if (f4width(ref) >= MAX_GET_WIDTH) continue;
1039
1040         w = f4width(ref);
1041         if (w <= 10) w = 10;
1042
1043         if (c > 2 && c < (rc - w))
1044         {
1045             w4(ur + 1, c - 2, vt);
1046             for (i = ur + 2; i < lr - 1; i++) w4(i, c - 2, vl);
1047         }
1048
1049         next_c = c + w + 2;
1050         if (next_c >= w4width(-1)) break;
1051
1052         w4(1, c, f4name(ref));
1053         g4field(-1, c, ref);
1054     }
1055
1056     return (w4select(-1));
1057 }
1058
1059 int  b4quick_edit()
1060 {
1061     int  r, j;
1062     long ref;
1063
1064     w4define(1, 0, 24, 79);
1065     w4attribute(KC);
1066     w4(0, 1, d4name());
1067
1068     w4memory();
1069     w4activate(-1);
1070     g4release(0);
1071
1072     r = 1;
1073     for (j = 1; j <= f4num_fields(); j++)
1074     {
1075         ref = f4j_ref(j);
1076         if (f4type(ref) == 'M') continue;
1077         if (f4width(ref) >= MAX_GET_WIDTH) continue;
1078
1079         if (r >= w4height(-1) - 1) break;
1080         w4(r, 2, f4name(ref));
1081         g4field(r, 14, ref);
1082         if (f4width(ref) > 64) g4width(f4width(ref), 64);
1083         r++;
1084     }
1085
1086     return (w4select(-1));
1087 }
```

```
/* u4error.c   modified for RMRS and DATA WINDOWS

MDS 11-21-91
see original code base version of u4error.c to port to UNIX

*/

#include <stdarg.h>
#include "rmrserr.h"
#include "p4misc.h"
#include "dw.h"
#include "d4all.h"
#include "rmrs.h"

HWND errwin;

int  v4error = -1;

typedef struct error_data_st
{
    int   error_num;
    char * error_data;
}
    ERROR_DATA;

ERROR_DATA error_data[] =
        {
            /* General Disk Access Errors */
        {
                E_CREATE, "Creating File"
            },
        {
                E_OPEN, "Opening File"
            },
        {
                E_READ, "Reading from File"
            },
        {
                E_LSEEK, "Seeking to File Position"
            },
        {
                E_WRITE, "Writing to File"
            },
        {
                E_CLOSE, "Closing File"
            },
        {
                E_REMOVE, "Removing File"
            },

            /* Database Specific Errors */
        {
                E_BAD_DBF, "File is not a Database:"
            },
        {
                E_D_MISSING, "No Open Database"
            },
        {
                E_REC_LENGTH, "Record Length is Too Large"
            },
        {
                E_FIELD, "Unrecognized Field"
            },

            /* Index File Specific Errors */
        {
                E_INDEX, "Building Index File"
            },
        {
                E_I_CLOSE, "Closing Index File"
            },
        {
                E_BAD_NDX, "File is not an Index File"
            },
```

```
46          {
46              E_I_DATE, "Index File is out of Date"
46          },
47          {
47              E_I_RECORD, "Index File Record does not Exist"
47          },
48          {
48              E_UNIQUE, "Key is not Unique"
48          },
49          {
49              E_I_TYPE, "Key Evaluates to Logical Result"
49          },
50          {
50              E_I_CHANGED, "Key Length or Type has Changed"
50          },
51          {
51              E_KEY_LEN, "Key Length over 100 Characters"
51          },
52          {
52              E_NO_INDEX, "Seek on Database with no Index File"
52          },
53          {
53              E_NUM_PARMS, "Wrong Number of Parameters in Expression"
53          },
54
55          /* Multi-User Errors */
56          {
56              E_LOCK, "Locking a File"
56          },
57          {
57              E_UNLOCK, "Unlocking a File"
57          },
58
59          /* Expression Evaluation Errors */
60          {
60              E_BASE_NAME, "Database not Located while Evaluating Expression"
60          },
61          {
61              E_COMPILE_NULL, "Executing Null Expression"
61          },
62          {
62              E_EXPECT, "Expecting \",\" or \")\" while Evaluating Expression"
62          },
63          {
63              E_COMPLETE, "Expression is not Complete"
63          },
64          {
64              E_DATE, "Illegal Date"
64          },
65          {
65              E_OVERFLOW, "Overflow while Evaluating Expression"
65          },
66          {
66              E_TYPE, "Parameter or Operator has the Wrong Type"
66          },
67          {
67              E_RIGHT, "Right Bracket Missing in Expression"
67          },
68          {
68              E_FUNCTION, "Unrecognized Function in Expression"
68          },
69          {
69              E_OPERATOR, "Unrecognized Operator in Expression"
69          },
70          {
70              E_VALUE, "Unrecognized Value in Expression"
70          }
70          ,
71          {
71              E_STRING_LONG, "Unterminated String in Expression"
71          }
71          ,
72
73          /* Memo File Errors */
```

```
74          {
74              E_EDITOR, "Editing Memo File with Editor"
74          }
74          ,
75          {
75              E_MEMO_NAME, "Memo File Name Inconsistency"
75          }
75          ,
76          {
76              E_MEMO_SIZE, "Memo File Entry is over 32767 Bytes"
76          },
77
78          /* Windowing and Menuing Errors */
79          {
79              E_WINDOW_REF, "Illegal Window Reference Number"
79          },
80
81          /* Extended Routine Errors */
82          {
82              E_RELATING, "Relating Databases"
82          }
82          ,
83          {
83              E_CONTROL, "No Controlling Database"
83          }
83          ,
84          {
84              E_RELATED, "Illegal Related Database"
84          }
84          ,
85
86          /* Memory Error */
87          {
87              E_MEMORY, "Out of Memory"
87          }
87          ,
88          {
88              E_ALLOCATE, "Memory Allocation Error"
88          }
88          ,
89
90          /* Internal Error */
91          {
91              E_INTERNAL, "Overwritten Memory"
91          }
91          ,
92
93          /* Sorting Errors */
94          {
94              E_SORT, "Not Enough Memory to Sort"
94          }
94          ,
95          {
95              E_SORT_ADD, "Too Many Records in Sort"
95          },
96      };
97
98  static int col, row;
99
100 void  error_out(char *);
101 void  error_out(char * ptr)
102 {
103     row++;
104     vatputs(errwin, row, col, ptr);
105 }
106
107 u4error(int error_num, char * msg, ...)
108 {
109     int      keyboard, i;
110     va_list  arg_marker;
111     char * ptr;
112     char  buffer[40];
113     row = 0;
114     col = 2;
```

```
115
116     errwin = vcreat(15, 40, ERR, YES);
117     vlocate(errwin, 5, 20);
118     vframe(errwin, ERR, FRDOUBLE);
119     vshadow(errwin, CURRENT, SHADOW75, BOTTOMRIGHT);
120     visible(errwin, YES, YES);
121     vmtitle(errwin, _TOP, CENTERJUST, ERR, " ERROR ");
122
123     v4error = error_num;
124
125     error_out("Error Number: ");
126     c4ltoa((long) error_num, buffer, 10);
127     buffer[10] = '\000';
128     error_out(buffer);
129     error_out("");
130
131     for (i = 0; i < sizeof (error_data) / sizeof (ERROR_DATA); i++)
132         if (error_data[i].error_num == error_num)
133         {
134             error_out(error_data[i].error_data);
135             break;
136         }
137
138     va_start(arg_marker, msg);
139     ptr = msg;
140
141     for (; ptr != (char *) 0; ptr = va_arg(arg_marker, char *))
142     {
143         error_out(ptr);
144     }
145
146     error_out("Press a key ...");
147     keyboard = getch();
148
149     if (error_num == E_MEMORY || error_num == E_INTERNAL || error_num == E_ALLOCATE)
150     {
151         vdelete(errwin, NONE);
152         exit(1);
153     }
154
155     vdelete(errwin, NONE);
156     return (keyboard);
157 }
```

```
 1 /*
 2  *  rmrsinit.c
 3  *
 4 */
 5
 6 #define SYSTDSCRP "System commands and description"
 7 #define SABTDSCRP "Show version "
 8 #define SDOSDSCRP "Exit to DOS temporarily (typing EXIT will return to RMRS)"
 9 #define SQUTDSCRP "Quit the RMRS program "
10 #define INITDSCRP "Change setup or initialize new AB supply files"
11 #define PROCDSCRP "Check repairs for resource availablity conflicts"
12 #define SCHDDSCRP "Schedule the repairs that are POSSIBLE"
13 #define CHNGDSCRP "Change the status of a repair"
14 #define GREPDSCRP "Generate a report"
15 #define QUITDSCRP "Exit program"
16 #define ILNSDSCRP "Load a new AB resource supply file from disk"
17 #define ILEQDSCRP "Load a new AB equipment supply file from disk"
18 #define ILMTDSCRP "Load a new AB materials supply file from disk"
19 #define PAUTDSCRP "Process ALL new PDES facility repair files in priority order"
20 #define PSELDSCRP "Select a single PDES facility repair file for processing"
21 #define PCOMDSCRP "Select a SUSPENDED repair and reattempt a resource compromise"
22 #define SFULDSCRP "Schedule all repairs (i.e. new AND previously scheduled repairs)"
23 #define SINCDSCRP "Schedule only newly processed repairs"
24 #define SPRIDSCRP "Set the scheduling order by adjusting repair priority"
25 #define CCOMDSCRP "Change the status of a QUEUED repair to COMPLETED"
26 #define CCANDSCRP "Change the status of a POSSIBLE or QUEUED repair to CANCELED"
27 #define GGANDSCRP "Generate a gantt chart"
28 #define GSCHDSCRP "Generate a tabular schedule listing"
29 #define ED        "Edit RMRS data files"
30
31 #include <stdio.h>
32 #include <string.h>
33 #include "color.h"
34 #include "dw.h"
35 #include "dwmenu.h"
36 #include "dwsystem.h"
37
38 #include "itemfns.h"
39 #include "rmrs.h"
40
41 HWND * mmain;
42
43 void globhelp(MENUITEM * item);
44 void IntroScreen(void);
45 void SetupMenu(void);
46
47 void SetupMenu(void)
48 {
49     HWND F1HelpWin;
50
51     MENUITEM * System;
52     MENUITEM * SystAbot, * SystXDOS, * SystCOMM, * SystQuit;
53
54     MENUITEM * EdFiles;
55     MENUITEM * EdFlsRep, * EdFlsEqSup, * EdFlsEqReq, * EdFlsMaSup, * EdFlsMaReq;
56     MENUITEM * EdFlsReset, * EdFlsRepInfo, * EdFlsFacPrty;
57
58     MENUITEM * InitSetup;
59     MENUITEM * InitNwFi, * InitMatl, * InitEqui;
60
61     MENUITEM * Process;
62     MENUITEM * ProcAuto, * ProcFacl, * ProcComp;
63
64     MENUITEM * Schedule;
65     MENUITEM * SchdFull, * SchdInc, * SchdPrior;
66
67     MENUITEM * ChangeStatus;
68     MENUITEM * ChngComp, * ChngCanc;
69
70     MENUITEM * GenerateReport;
71     MENUITEM * GnRpSum, * GnRpDet, * GnRpGant;
72
73     /*** set up help micro-window "F1 - HELP" ***/
74     F1HelpWin = vcreat(1, 80, EMPHNORML, NO);
```

```
    vlocate(F1HelpWin, 24, 0);
    vratputs(F1HelpWin, 0, 0, RED_ON_WHITE, " F1 ");
    vatputs(F1HelpWin, 0, 4, "Help 3 ");
    visible(F1HelpWin, YES, NO);

    /*** define main menu ***/
    mmain = MNUCreateHdr(PULLDOWN);
    MNUSetGlobalHelp(mmain, globhelp);
    MNUSetShadow(mmain, SHADOW75, BOTTOMRIGHT);
    MNUShowHotKey(mmain, YES);
    MNUSetDescPosition(mmain, 24, 11);
    MNUSetDownIndicator(mmain, NO);
    MNUSetSpaces(mmain, 2);
    MNUSetDropDown(mmain, YES);
    MNUSetAttributes(mmain, EMPHNORML, EMPHNORML, EMPHNORML, REVNORML/* HELP*/, HIGHNORML,
        EMPHNORML, REVERR, EMPHNORML);

    /*** define menu bar items ***/
    System = MNUAddItem(" System ", SYSTDSCRP, 'y', ALTY, NULL, mmain, NULLF);
    InitSetup = MNUAddItem(" Init/Setup ", INITDSCRP, 'I', ALTI, NULL, mmain, NULLF);
    Process = MNUAddItem(" Process ", PROCDSCRP, 'P', ALTP, NULL, mmain, NULLF);
    Schedule = MNUAddItem(" Schedule ", SCHDDSCRP, 'S', ALTS, NULL, mmain, NULLF);
    ChangeStatus = MNUAddItem(" Change Status ", CHNGDSCRP, 'C', ALTC, NULL, mmain, NULLF);
    GenerateReport = MNUAddItem(" Gen Report ", GREPDSCRP, 'R', ALTR, NULL, mmain, NULLF);

    /*** set up pull down menus ***/
    SystAbot = MNUAddItem("About", SABTDSCRP, ' ', 0, System, mmain, SystAbotFn);

    EdFiles = MNUAddItem("Edit Data", ED, ' ', ALTE, System, mmain, NULLF);
    EdFlsRep = MNUAddItem("Repair File", ED, ' ', 0, EdFiles, mmain, EdFlsRepFn);
    EdFlsEqSup = MNUAddItem("Equip Supply", ED, ' ', 0, EdFiles, mmain, EdFlsEqSupFn);
    EdFlsEqReq = MNUAddItem("Equip Required", ED, ' ', 0, EdFiles, mmain, EdFlsEqReqFn);
    EdFlsMaSup = MNUAddItem("Material Supply", ED, ' ', 0, EdFiles, mmain, EdFlsMaSupFn);
    EdFlsMaReq = MNUAddItem("Material Required", ED, ' ', 0, EdFiles, mmain, EdFlsMaReqFn);
    EdFlsRepInfo = MNUAddItem("Repair Info", ED, ' ', 0, EdFiles, mmain, EdFlsRepInfoFn);
    EdFlsFacPrty = MNUAddItem("Facility Priority", ED, ' ', 0, EdFiles, mmain, EdFlsFacPrtyFn);
    EdFlsReset = MNUAddItem("Reset All Data", ED, ' ', 0, EdFiles, mmain, EdFlsResetFn);

    SystXDOS = MNUAddItem("DOS shell", SDOSDSCRP, ' ', ALTD, System, mmain, SystXDOSFn);

    MNUSetSeparatorBefore(SystXDOS);

    SystQuit = MNUAddItem("Quit", SQUTDSCRP, ' ', ALTQ, System, mmain, SystQuitFn);

    InitNwFi = MNUAddItem("Load new AB supply file", ILNSDSCRP, ' ', 0, InitSetup, mmain, NULLF
);
    InitMatl = MNUAddItem("AB materials supply file", ILMTDSCRP, ' ', 0, InitNwFi, mmain, InitM
atlFn);
    InitEqui = MNUAddItem("AB equipment supply file", ILEQDSCRP, ' ', 0, InitNwFi, mmain, InitE
quiFn);

    ProcAuto = MNUAddItem("Auto Mode", PAUTDSCRP, ' ', 0, Process, mmain, ProcAutoFn);
    ProcFacl = MNUAddItem("Select facility", PSELDSCRP, ' ', 0, Process, mmain, ProcFaclFn);
    ProcComp = MNUAddItem("Compromise Retry", PCOMDSCRP, ' ', 0, Process, mmain, ProcCompFn);

    SchdFull = MNUAddItem("Full Schedule", SFULDSCRP, ' ', 0, Schedule, mmain, SchdFullFn);
    SchdInc = MNUAddItem("Update Schedule", SINCDSCRP, ' ', 0, Schedule, mmain, SchdIncFn);
    SchdPrior = MNUAddItem("Set Priority", SPRIDSCRP, ' ', 0, Schedule, mmain, SchdPriorFn);

    ChngComp = MNUAddItem("Completed", CCOMDSCRP, ' ', 0, ChangeStatus, mmain, ChngCompFn);
    ChngCanc = MNUAddItem("Canceled", CCANDSCRP, ' ', 0, ChangeStatus, mmain, ChngCancFn);

    GnRpSum = MNUAddItem("Summary Schedule", GSCHDSCRP, ' ', 0, GenerateReport, mmain, GnRpSchd
Fn);
    GnRpDet = MNUAddItem("Detailed Schedule", GSCHDSCRP, ' ', 0, GenerateReport, mmain, GnRpSch
dFn);
    GnRpGant = MNUAddItem("Gantt Chart", GGANDSCRP, ' ', 0, GenerateReport, mmain, GnRpGantFn);

}

#define HELPFILENAME "RMRS.HLP"
#define HWLR  25
#define HWLC  80
#define HWPR   7
```

```
144 #define HWPC  40
145 #define HWROW  9
146 #define HWCOL 20
147
148 void globhelp(MENUITEM * item)
149 {
150     HWND hwinptr;
151     char * helpstr;
152
153     /* DEFINE HELP WINDOW */
154     hwinptr = vcreat(HWLR, HWLC, HELP, NO);
155     vwind(hwinptr, HWPR, HWPC, 0, 0);
156     vlocate(hwinptr, HWROW, HWCOL);
157     vframe(hwinptr, REVHIGHHELP, FRSINGLE);
158     vmtitle(hwinptr, _TOP, CENTERJUST, REVHIGHHELP, " RMRS Help (ESC to quit) ");
159     visible(hwinptr, YES, YES);
160
161     helpstr = ExtractFromHelpFile(HELPFILENAME,
162         (item->strtoggle ? item->item1 : item->item0));
163     if (helpstr)
164     {
165         vdispstr(hwinptr, helpstr);
166         scroll_win_keys(hwinptr, HWLR, HWLC, HWPR, HWPC);
167     }
168     else
169     {
170         /* NO HELP FILE */
171         vatputs(hwinptr, 2, 1, "<Cannot find help file '");
172         vputs(hwinptr, HELPFILENAME);
173         vputs(hwinptr, "'>");
174         getch();
175     }
176
177     /* DELETE HELP WINDOW */
178     vdelete(hwinptr, NONE);
179 }
180
181 void IntroScreen(void)
182 {
183     HWND IntroScreen;
184
185     /*** set background ***/
186     pclrchar(SHADOW25);
187
188     /*** display intro screen ***/
189     IntroScreen = vcreat(17, 70, REVNORML, YES);
190     vlocate(IntroScreen, 3, 5);
191     vshadow(IntroScreen, NULL, SHADOW75, BOTTOMRIGHT);
192     vframe(IntroScreen, REVNORML, FRDOUBLE);
193     vatputas(IntroScreen, 1, 8, HELP, "  Automated RESOURCE MANAGER / REPAIR SCHEDULER (RMRS)")
;
194     vatputs(IntroScreen, 2, 9, "Automated RESOURCE MANAGER / REPAIR SCHEDULER (RMRS)");
195     vatputas(IntroScreen, 3, 8, HELP, "  Automated RESOURCE MANAGER / REPAIR SCHEDULER (RMRS)")
;
196     vatputs(IntroScreen, 5, 9, "                    for the                         ");
197     vatputs(IntroScreen, 7, 9, "   Post-Attack Facility Damage Assessment System    ");
198     vatputs(IntroScreen, 9, 9, "              <<< POST - DAM >>>                    ");
199     vatputs(IntroScreen, 10, 28, "Version ");
200     vputs(IntroScreen, VERSION);
201     vputs(IntroScreen, ".");
202     vputs(IntroScreen, REVISION);
203     vatputs(IntroScreen, 13, 9, "          AFCESA/RACS Tyndall AFB, FL");
204     vatputs(IntroScreen, 14, 9, "        Applied Research Associates, Inc.");
205     vatputs(IntroScreen, 16, 9, "          Hit any key to continue ...      ");
206     visible(IntroScreen, YES, YES);
207
208     /*** wait for keystroke and return ***/
209     getkey();
210     vdelete(IntroScreen, NONE);
211     return;
212 }
```

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <dir.h>
#include <time.h>
#include <string.h>
#include <alloc.h>
#include "d4base.h"
#include "dw.h"
#include "dwmenu.h"
#include "dwsystem.h"
#include "rmrsutil.h"
#include "rmrs.h"

#define BUFFLENGTH 300

int ReprtSched(void)
{
    int status, elenum, cnt = 0;
    int r = 0, c = 1, key, lwinr, lwinc, pwinr, pwinc;
    long repid, facnum;
    time_t start, dur, end, currtime;
    char eledescstr[53], dammodestr[53], repstgystr[53], schedstatus, * timeptr;
    char facfuncstr[53];
    HWND win;

    lwinr = 200;
    lwinc = 78;
    pwinr = 21;
    pwinc = 78;
    /* WINDOW STUFF */
    win = vcreat(lwinr, lwinc, bldattr(LIGHTBLUE, 0), YES);
    if (win < 1) u4error(-1, "Error creating window",(char *) 0);
    vwind(win, pwinr, pwinc, 0, 0);
    vlocate(win, 2, 1);
    vframe(win, REVNORML, FRDOUBLE);
    vmtitle(win, _TOP, CENTERJUST, REVNORML, " Report - Repair Schedule ");

    /* OPEN DBF AND INDEX FILES */
    if (open_eqpreq(1))
    {
        u4error(-1, "BAD RETURN CODE", "OPEN_EQPREQ",(char *) 0);
        vdelete(win, NONE);
        return (-1);
    }
    if (open_repinfo(1))
    {
        u4error(-2, "BAD RETURN CODE", "OPEN_REPINFO",(char *) 0);
        vdelete(win, NONE);
        return (-2);
    }
    if (open_repair(1))
    {
        u4error(-3, "BAD RETURN CODE", "OPEN_REPAIR",(char *) 0);
        vdelete(win, NONE);
        return (-3);
    }

    d4select(eqpreq_dbf);
    i4select(eqpreq_ndx2);
    status = x4seek_str("REPAIR TEAM");
    switch (status)
    {
    case (0) :
    case (1) :
        while ((! d4eof()) &&(is_repteam()))
        {
            switch (f4char(f4ref("STATUS")))
            {
            case ('A') :
            case ('L') :
                repid = f4long(f4ref("REPID"));
                start = f4long(f4ref("START"));
```

```
                dur = f4long(f4ref("DURATION"));
                schedstatus = f4char(f4ref("STATUS"));
                end = (start + dur);

                d4select(repair_dbf);
                i4select(repair_ndx1);
                x4seek_double(repid);
                facnum = f4long(f4ref("FACNUM"));

                d4select(repinfo_dbf);
                i4select(repinfo_ndx1);
                x4seek_double(repid);
                elenum = f4int(f4ref("ELENUM"));
                strcpy(eledescstr, f4str(f4ref("ELEDESC")));
                strcpy(facfuncstr, f4str(f4ref("FACFUNC")));
                strcpy(dammodestr, f4str(f4ref("DAMMODE")));
                strcpy(repstgystr, f4str(f4ref("REPSTGY")));

                if ((cnt == 0))
                {
                    vatputs(win, r++, c, "Schedule Listing");
                    currtime = time(NULL);
                    timeptr = ctime(& currtime);
                    timeptr[strlen(timeptr) - 1] = '\0';
                    vatputf(win, r++, c, "Generated : %s", timeptr);
                    vatputf(win, r++, c, "-------------------------------------------------------
-----------");
                }
                d4select(eqpreq_dbf);
                r++;
                vatputf(win, r++, c, "Repair ID    :   %-19d    ", repid);
                timeptr = ctime(& start);
                timeptr[strlen(timeptr) - 1] = '\0';
                vatputf(win, r++, c, "START TIME   :   %s", timeptr);
                timeptr = ctime(& end);
                timeptr[strlen(timeptr) - 1] = '\0';
                vatputf(win, r++, c, "END TIME     :   %s", timeptr);
                vatputf(win, r++, c, "facility Num :   B%-ld", facnum);
                vatputf(win, r++, c, "facility Func:   %-s", facfuncstr);
                vatputf(win, r++, c, "element #    :   %d", elenum);
                vatputf(win, r, c, "elem desc    : %-19s", eledescstr);
                vatputf(win, r++, c + 60, "SCHD STATUS: %c", schedstatus);
                vatputf(win, r++, c, "damage mode  : %s", dammodestr);
                vatputf(win, r++, c, "rep strategy : %s", repstgystr);
                cnt++;
                break;
            default :;                  /* do nothing                                   */
            }
            d4select(eqpreq_dbf);
            d4skip(1);

        }
        d4close_all();
        visible(win, YES, YES);
        scroll_win_keys(win, lwinr, lwinc, pwinr, pwinc);
        cnt = vfputwin(win, "SCHED.OUT", 1);
        if (cnt != win) u4error(cnt, "DID NOT WRITE FILE", (char *) 0);
        vdelete(win, NONE);
        break;
    case (2) :
    case (3) :
    default :;                          /* no repairs in EQPREQ.DBF                     */
        u4error(0, "REPORT GENERATION ERROR",(char *) 0);
    }
    return (1);
}

int is_repteam(void)
{

    if (strstr(f4str(f4ref("EQPDESC")), "repair team") != NULL)
        return (1);
    else
        return (0);
```

```
148 }
149
150 /* ReprtGant
151 *
152 *
153 */
154 int ReprtGantt(void)
155 {
156     int done, i, thisone, r = 0, c = 1;
157     char status, linebuff[BUFFLENGTH], curdesc[26], * tmpdesc, * timeptr;
158     long start, duration, currepid, minstart, delta = 900L, curtime, maxend;
159     long cureqpid, eqpidfld, eqpdescfld, eqpid;
160     long statusfld, startfld, durationfld, repidfld, repid;
161     HWND win;
162     unsigned int key = 0;
163     int lwinr, lwinc, pwinr, pwinc;
164
165     /* WINDOW STUFF */
166     lwinr = 100;                        /* logical window rows                  */
167     lwinc = 300;                        /* logical window columns               */
168     pwinr = 21;                         /* physical window rows                 */
169     pwinc = 78;                         /* physical window columns              */
170     win = vcreat(lwinr, lwinc, bldattr(LIGHTBLUE, 0), YES);
171     vwind(win, pwinr, pwinc, 0, 0);
172     vlocate(win, 2, 1);
173     vframe(win, REVNORML, FRDOUBLE);
174     vmtitle(win, _TOP, CENTERJUST, REVNORML, " Report - Gantt Chart ");
175
176     /* OPEN DBF AND INDEX FILES */
177     if (open_eqpreq(1))
178     {
179         u4error(-1, "BAD RETURN CODE", "OPEN_EQPREQ",(char *) 0);
180         vdelete(win, NONE);
181         return (-1);
182     }
183     i4select(eqpreq_ndx4);              /* key = START                          */
184
185     statusfld = f4ref("STATUS");
186     startfld = f4ref("START");
187     durationfld = f4ref("DURATION");
188     repidfld = f4ref("REPID");
189     eqpidfld = f4ref("EQPID");
190     eqpdescfld = f4ref("EQPDESC");
191
192     /**** GET MINSTART & MAXEND ****/
193     minstart = 2147000000;
194     maxend = 0L;
195     x4top();
196     while (! d4eof())
197     {
198         status = f4char(statusfld);
199         start = f4long(startfld);
200         duration = f4long(durationfld);
201         if ((status == 'A' || status == 'L') && start < minstart)
202             minstart = start;
203         if ((status == 'A' || status == 'L') && start + duration > maxend)
204             maxend = start + duration;
205         x4skip(1L);
206     }
207
208     /* BUILD SCHEDULE */
209     curtime = time(NULL);
210     timeptr = ctime(& curtime);
211     timeptr[strlen(timeptr) - 1] = '\0';
212     vatputf(win, r++, c, "Schedule as of :%s", timeptr);
213     vatputf(win, r++, c, "Each character represents %ld minutes", delta / 60);
214     vatputf(win, r++, c, "Total repair time is %.2f hours", ((float) (maxend - minstart)) / 360
0.);
215     vatputf(win, r++, c, "Repair Schedule");
216     vatputf(win, r++, c, "%20s|", " ");
217     x4top();
218     while (! d4eof())
219     {
220         for (i = 0; i < 299; i++) linebuff[i] = 32;
```

```
221         linebuff[299] = '\0';
222         status = f4char(statusfld);
223         while (! (status == 'A' || status == 'L'))
224         {
225             x4skip(1L);
226             status = f4char(statusfld);
227         }
228         currepid = f4long(repidfld);
229         start = f4long(startfld);
230         duration = f4long(durationfld);
231         Add2Sch(linebuff, minstart, start, duration, delta, status, currepid);
232         TrimSch(linebuff);
233         vatputf(win, r++, c, "REPID %-4ld%10s:%s", currepid, " ", linebuff);
234         while (currepid == f4long(repidfld))
235         {
236             x4skip(1L);
237         }
238     }
239
240     i4select(eqpreq_ndx5);
241     r++;
242     vatputf(win, r++, c, "Equipment Allocation Schedule");
243     vatputf(win, r++, c, "%20s|", " ");
244     x4top();
245     while (! d4eof())
246     {
247         for (i = 0; i < 299; i++) linebuff[i] = 32;
248         linebuff[299] = '\0';
249         cureqpid = f4long(eqpidfld);
250         strcpy(curdesc, f4str(eqpdescfld));
251         strupr(curdesc);
252         c4trim_n(curdesc, 25);
253         eqpid = f4long(eqpidfld);
254         thisone = 0;
255         while (cureqpid == eqpid && ! d4eof())
256         {
257             status = f4char(statusfld);
258             if ((status == 'A' || status == 'L' || status == 'C'))
259             {
260                 thisone = 1;
261                 start = f4long(startfld);
262                 duration = f4long(durationfld);
263                 tmpdesc = f4str(eqpdescfld);
264                 strupr(tmpdesc);
265                 c4trim_n(tmpdesc, 255);
266                 repid = f4long(f4ref("REPID"));
267                 Add2Sch(linebuff, minstart, start, duration, delta, status, repid);
268             }
269             x4skip(1L);
270             eqpid = f4long(eqpidfld);
271         }
272         TrimSch(linebuff);
273         if (thisone)
274             vatputf(win, r++, c, "%-12s ID %-4ld:%s", curdesc, cureqpid, linebuff);
275     }
276     d4close_all();
277     visible(win, YES, YES);
278     scroll_win_keys(win, lwinr, lwinc, pwinr, pwinc);
279     vfputwin(win, "GANTT.OUT", 1);
280     vdelete(win, NONE);
281     return (0);
282 }
283
284 static int TrimSch(char * linebuff)
285 {
286     int i;
287     for (i = 298; i > -1; i--)
288     {
289         if (linebuff[i] != 32 || i == 0)
290         {
291             if (i) linebuff[i + 1] = '\0';
292             else linebuff[0] = '\0';
293             break;
294         }
```

```
295     }
296 }
297
298 static int Add2Sch(char * linebuff, long minstart, long start, long duration,
299                 long delta, char marker, long repid)
300 {
301     double dbeg, dend;
302     int beg, end, i, buffstart, buffend;
303
304     dbeg = ((double) (start - minstart)) / ((double) delta);
305     dend = dbeg + ((double) duration / ((double) delta));
306     dbeg = (((ceil(dbeg) - dbeg) < .5) ? ceil(dbeg) : floor(dbeg));
307     dend = (((ceil(dend) - dend) < .5) ? ceil(dend) : floor(dend));
308
309     beg = (int) dbeg;
310     end = (int) dend - 1;
311
312     buffstart = 0;
313     buffend = 298;
314
315     if (beg < 0 || beg > 298) return (-1);
316     if (end > 298) end = 298;
317     for (i = beg; i <= end; i++) linebuff[i] = (char) (repid + 64);
318     return (1);
319 }
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dir.h>
#include <time.h>
#include <string.h>
#include <alloc.h>
#include "d4base.h"
#include "dw.h"
#include "dwmenu.h"
#include "dwsystem.h"
#include "rmrs.h"


typedef struct
{
    int dbf;
    int ndx;
    long rec;
}
    CONTEXT;

typedef struct llll
{
    long val;
    struct llll *next;
}
    LLLL;

int CountLLLLNodes(LLLL * ptr);
void context_save(CONTEXT *);
void context_restore(CONTEXT *);
LLLL * CreateLLLLNode(void);
int scroll_win_keys(int, int, int, int, int);
int InLLLL(LLLL * ptr, long);
int DumpLLLLNodes(LLLL * ptr, FILE * deblog, char * spaces);
LLLL * FreeLLLL(LLLL * ptr);

void context_save(CONTEXT * def)
{
    def->rec = d4recno();
    def->ndx = i4select(-1);
    def->dbf = d4select(-1);
}

void context_restore(CONTEXT * def)
{
    if (def->dbf > -1)
    {
        d4select(def->dbf);
        i4select(def->ndx);
        if (d4eof()) d4skip(-1L);
        if (d4bof()) d4skip(1L);
        d4go(def->rec);
    }
}

int scroll_win_keys(HWND win, int lwinr, int lwinc, int pwinr, int pwinc)
{
    unsigned int key = 13;
    while (key != ESC)
    {
        switch (key = getkey())
        {
        case CTRLCURRT :
            if (vmovedir(win, 0, pwinc) != DWSUCCESS)
                vloc(win, 0, 0);
            break;
        case CTRLCURLF :
            if (vmovedir(win, 0, - pwinc) != DWSUCCESS)
                vloc(win, 0, 0);
        case CURUP :
            vmovedir(win, -1, 0);
            break;
```

```
70          case CURDN :
71              vmovedir(win, 1, 0);
72              break;
73          case CURLF :
74              vmovedir(win, 0, -1);
75              break;
76          case CURRT :
77              vmovedir(win, 0, 1);
78              break;
79          case CTRLUP :
80              if (vmovedir(win, - pwinr, 0) != DWSUCCESS)
81                  vloc(win, 0, 0);
82              break;
83          case PGUP :
84              if (vmovedir(win, - pwinr, 0) != DWSUCCESS)
85                  vloc(win, 0, 0);
86              break;
87          case CTRLCURDN :
88              if (vmovedir(win, pwinr, 0) != DWSUCCESS)
89                  vloc(win, lwinr - pwinr, 0);
90              break;
91          case PGDN :
92              if (vmovedir(win, pwinr, 0) != DWSUCCESS)
93                  vloc(win, lwinr - pwinr, 0);
94              break;
95          case HOME :
96              vloc(win, 0, 0);
97              break;
98          case ENDKEY :
99              vloc(win, lwinr - pwinr, 0);
100             break;
101         default :
102             break;
103         }
104     }
105     return (0);
106 }
107
108 int InLLLL(LLLL * ptr, long val)
109 {
110     if (ptr == NULL) return (0);
111
112     if (val == ptr->val) return (1);
113
114     if (ptr->val == -1000000L)
115     {
116         ptr->val = val;
117         return (0);
118     }
119
120     while (ptr->next)
121     {
122         ptr = ptr->next;
123         if (val == ptr->val) return (1);
124     }
125     ptr->next = CreateLLLLNode();
126     ptr = ptr->next;
127     ptr->val = val;
128     return (0);
129 }
130
131
132 LLLL * CreateLLLLNode()
133 {
134     LLLL * ptr;
135     ptr = (LLLL *) malloc(sizeof (LLLL));
136     ptr->val = -1000000L;
137     ptr->next = NULL;
138     return (ptr);
139 }
140
141 int CountLLLLNodes(LLLL * ptr)
142 {
143     int i = 0;
```

```
144     if (ptr == NULL)
145     {
146         return (i);
147     }
148     while (ptr->next)
149     {
150         i++;
151         ptr = ptr->next;
152     }
153     i++;
154     return (i);
155 }
156
157 int DumpLLLLNodes(LLLL * ptr, FILE * deblog, char * spaces)
158 {
159     int i = 0;
160     if (ptr == NULL)
161     {
162         return (i);
163     }
164     while (ptr->next)
165     {
166         i++;
167         fprintf(deblog, "%snode %d: %ld\n", spaces, i, ptr->val);
168         ptr = ptr->next;
169     }
170     i++;
171     fprintf(deblog, "%snode %d: %ld\n", spaces, i, ptr->val);
172     return (i);
173 }
174
175 LLLL * FreeLLLL(LLLL * ptr)
176 {
177     LLLL * tmp;
178
179     while (ptr->next != NULL)
180     {
181         tmp = ptr;
182         ptr = ptr->next;
183         free(tmp);
184     }
185     free(ptr);
186     return (NULL);
187 }
188
189 /*  FUNCTIONS TO OPEN DBF FILES AND ATTACH ALL INDEX FILES
190  *
191  *  DBFfname     NDXREF          NDXfname     KEY
192  *  ------------------------------------------------------------------------
193  *  repair       repair_ndx1     reprndx1     REPID
194  *               repair_ndx2     reprndx2     upper(STATUS)+str(PRIORITY,2,0)
195  *                                            +str(FACNUM,4,0)
196  *
197  *  facprt       facprt_ndx1     facpndx1     FACNUM
198  *
199  *  eqpsup       eqpsup_ndx1     eqsundx1     upper(EQPDESC)
200  *
201  *  eqpreq       eqpreq_ndx1     eqrendx1     REPID
202  *               eqpreq_ndx2     eqrendx2     upper(EQPDESC)+str(START,10,0)
203  *               eqpreq_ndx3     eqrendx3     upper(EQPDESC)+str(START+DURATION,10,0)
204  *               eqpreq_ndx4     eqrendx4     START
205  *               eqpreq_ndx5     eqrendx5     upper(EQPDESC)+str(EQPID,10,0)
206  *
207  *  matsup       matsup_ndx1     masundx1    upper(MATDESC)
208  *
209  *  matreq       matreq_ndx1     marendx1    REPID
210  *               matreq_ndx2     marendx2    MATID
211  *
212  *  repinfo      repinfo_ndx1    repindx1    REPID
213  *
214 */
215
216 int  repair_dbf, repair_ndx1, repair_ndx2;
217 int  eqpsup_dbf, eqpsup_ndx1;
```

```
218 int  eqpreq_dbf, eqpreq_ndx1, eqpreq_ndx2, eqpreq_ndx3,
219      eqpreq_ndx4, eqpreq_ndx5;
220 int  matsup_dbf, matsup_ndx1;
221 int  matreq_dbf, matreq_ndx1, matreq_ndx2;
222 int  repinfo_dbf, repinfo_ndx1;
223 int  facprt_dbf, facprt_ndx1;
224
225 int open_repair(int rendx)
226 {
227     int status = 0;
228     repair_dbf = d4use_excl("REPAIR.DBF");
229     if (repair_dbf < 0) return (repair_dbf);
230     x4filter(d4deleted);
231     repair_ndx1 = i4open("REPRNDX1.NDX");
232     if (repair_ndx1 < 0) return (repair_ndx1);
233     repair_ndx2 = i4open("REPRNDX2.NDX");
234     if (repair_ndx2 < 0) return (repair_ndx2);
235
236     if (rendx) status = i4reindex(-1);
237     return status;
238 }
239
240 int open_eqpsup(int rendx)
241 {
242     int status = 0;
243     eqpsup_dbf = d4use_excl("eqpsup.DBF");
244     if (eqpsup_dbf < 0) return (eqpsup_dbf);
245     x4filter(d4deleted);
246     eqpsup_ndx1 = i4open("EQSUNDX1.NDX");
247     if (eqpsup_ndx1 < 0) return (eqpsup_ndx1);
248     if (rendx) status = i4reindex(-1);
249     return status;
250 }
251
252 int open_eqpreq(int rendx)
253 {
254     int status = 0;
255     eqpreq_dbf = d4use_excl("eqpreq.DBF");
256     if (eqpreq_dbf < 0) return (eqpreq_dbf);
257     x4filter(d4deleted);
258     eqpreq_ndx1 = i4open("EQRENDX1.NDX");
259     if (eqpreq_ndx1 < 0) return (eqpreq_ndx1);
260     eqpreq_ndx2 = i4open("EQRENDX2.NDX");
261     if (eqpreq_ndx2 < 0) return (eqpreq_ndx2);
262     eqpreq_ndx3 = i4open("EQRENDX3.NDX");
263     if (eqpreq_ndx3 < 0) return (eqpreq_ndx3);
264     eqpreq_ndx4 = i4open("EQRENDX4.NDX");
265     if (eqpreq_ndx4 < 0) return (eqpreq_ndx4);
266     eqpreq_ndx5 = i4open("EQRENDX5.NDX");
267     if (eqpreq_ndx5 < 0) return (eqpreq_ndx5);
268     if (rendx) status = i4reindex(-1);
269     return status;
270 }
271
272 int open_matsup(int rendx)
273 {
274     int status = 0;
275     matsup_dbf = d4use_excl("matsup.DBF");
276     if (matsup_dbf < 0) return (matsup_dbf);
277     x4filter(d4deleted);
278     matsup_ndx1 = i4open("MASUNDX1.NDX");
279     if (matsup_ndx1 < 0) return (matsup_ndx1);
280     if (rendx) status = i4reindex(-1);
281     return status;
282 }
283
284 int open_matreq(int rendx)
285 {
286     int status = 0;
287     matreq_dbf = d4use_excl("matreq.DBF");
288     if (matreq_dbf < 0) return (matreq_dbf);
289     x4filter(d4deleted);
290     matreq_ndx1 = i4open("MARENDX1.NDX");
291     if (matreq_ndx1 < 0) return (matreq_ndx1);
```

```
292     matreq_ndx2 = i4open("MARENDX2.NDX");
293     if (matreq_ndx2 < 0) return (matreq_ndx2);
294     if (rendx) status = i4reindex(-1);
295     return status;
296 }
297
298 int open_repinfo(int rendx)
299 {
300     int status = 0;
301     repinfo_dbf
302     = d4use_excl("repinfo.DBF");
303     if (repinfo_dbf < 0) return (repinfo_dbf);
304     x4filter(d4deleted);
305     repinfo_ndx1 = i4open("REPINDX1.NDX");
306     if (repinfo_ndx1 < 0) return (repinfo_ndx1);
307     if (rendx) status = i4reindex(-1);
308     return status;
309 }
310
311 int open_facprt(int rendx)
312 {
313     int status = 0;
314     facprt_dbf = d4use_excl("facprty.DBF");
315     if (facprt_dbf < 0) return (facprt_dbf);
316     x4filter(d4deleted);
317     eqpsup_ndx1 = i4open("FACPNDX1.NDX");
318     if (facprt_ndx1 < 0) return (facprt_ndx1);
319     if (rendx) status = i4reindex(-1);
320     return status;
321 }
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dir.h>
#include <time.h>
#include <string.h>
#include <alloc.h>
#include "rmrsutil.h"
#include "rmrs.h"
#include "d4base.h"
#include "dw.h"
#include "dwmenu.h"
#include "dwsystem.h"

long FindEarliest(char * eqpdesc, long duration, long repstart, long recno);
int SchedAuto(int resched);

static FILE * deblog;
static long defrepstart;

/* SchedAuto
*
*
*
*/
int SchedAuto(int resched)
{
    int stat, exitcode = 0, totalreprs = 0, complreprs = 0;
    char * statptr;
    long repid;
    extern HWND win;
    LLLL * replst = NULL, * rlptr = NULL;
    CONTEXT repair_ctx;

    /* SET MODULE GLOBALS */
    deblog = fopen("debug.log", "w");
    defrepstart = time(NULL) + 1800L;  /* CURRENT TIME + 30 MINUTES                    */

    /* WINDOW STUFF */
    win = vcreat(7, 40, EMPHNORML, YES);
    vautoshd(win, CURRENT, SHADOW75, BOTTOMRIGHT);
    vlocate(win, 9, 20);
    vframe(win, EMPHNORML, FRDOUBLE);
    if (resched) vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Full Schedule ");
    else vmtitle(win, _TOP, CENTERJUST, EMPHNORML, " Incremental Schedule ");
    visible(win, YES, YES);

    /* OPEN DBF AND INDEX FILES */
    vatputs(win, 1, 2, "Opening Data Files ...");
    if (open_eqpreq(1))
    {
        u4error(-1, "BAD RETURN CODE", "OPEN_EQPREQ",(char *) 0);
        vdelete(win, NONE);
        return (-1);
    }
    if (open_eqpsup(1))
    {
        u4error(-2, "BAD RETURN CODE", "OPEN_EQPSUP",(char *) 0);
        vdelete(win, NONE);
        return (-2);
    }
    if (open_repair(1))
    {
        u4error(-3, "BAD RETURN CODE", "OPEN_REPAIR",(char *) 0);
        vdelete(win, NONE);
        return (-3);
    }

    /* RESET STATUS FOR REPAIR=Q & EQPREQ=A TO P & N RESPECTIVELY IF resched */
    if (resched)
    {
        vatputs(win, 1, 2, "Clearing Schedule ...");
        d4select(repair_dbf);
        for (x4top(); ! d4eof(); x4skip(1L))
```

```
75              if (f4char(f4ref("STATUS")) == 'Q') f4r_char(f4ref("STATUS"), 'P');
76
77          d4select(eqpreq_dbf);
78          for (x4top(); ! d4eof(); x4skip(1L))
79              if (f4char(f4ref("STATUS")) == 'A') f4r_char(f4ref("STATUS"), 'N');
80      }
81
82      /* BUILD RECNO LIST FOR ELIGABLE REPAIRS */
83      d4select(repair_dbf);
84      i4select(repair_ndx2);
85      stat = x4seek_str("P");             /* find first POSSIBLE repair                    */
86      stat = ((stat == 1 || stat == 0) ? 1 : 0);
87      if (! stat)
88      {
89          u4error(-4, "No possible repairs found!",(char *) 0);
90          d4close_all();
91          vdelete(win, NONE);
92          return (-4);                    /* return no possible repairs found              */
93      }
94      replst = CreateLLLLNode();
95      while (stat)
96      {
97          InLLLL(replst, d4recno());
98          x4skip(1L);
99          statptr = f4str(f4ref("STATUS"));
100         if (statptr[0] == 'P' && ! d4eof()) stat = 1;
101         else stat = 0;
102     }
103     fprintf(deblog, "RECNO List of repairs to be scheduled\n");
104     DumpLLLLNodes(replst, deblog, "  ");
105     fprintf(deblog, "\n");
106     totalreprs = CountLLLLNodes(replst);
107
108     /* SCHEDULE EACH ELIGABLE REPAIR IN replst */
109     complreprs = 0;
110     rlptr = replst;
111     while (rlptr)
112     {
113         x4go(rlptr->val);
114         repid = f4long(f4ref("REPID"));
115         vatputf(win, 1, 2, "Processing REPID : %4ld", repid);
116         vatputf(win, 2, 2, "Repairs Processed: %4d", complreprs);
117         vatputf(win, 3, 2, "Percent Complete :  %3.0f%", ((float) complreprs / (float) totalrep
rs) * 100);
118         fprintf(deblog, "************************************************************\n");
119         fprintf(deblog, "\nBEGIN PROCESSING REPID: %4ld\n", repid);
120         if (CalcSchedule(repid))
121         {
122             exitcode = 1;
123             break;
124         }
125         f4r_char(f4ref("STATUS"), 'Q');
126         fprintf(deblog, "PROCESSING COMPLETE REPID: %4ld\n", repid);
127         rlptr = rlptr->next;
128         complreprs++;
129     }
130     fprintf(deblog, "************************************************************\n");
131     fprintf(deblog, "SCHEDULE PROCESSING COMPLETE\n");
132
133     /* DISPLAY FINAL STATS */
134     vatputf(win, 2, 2, "Repairs Processed:  %3d", complreprs);
135     vatputs(win, 3, 2, "Percent Complete :  100");
136     vatputs(win, 5, 2, "press any key ...");
137     getch();
138     /* FREE replst STORAGE */
139     if (replst) FreeLLLL(replst);
140     replst = NULL;
141     rlptr = NULL;
142     /* REMOVE PROGRESS WINDOW */
143     vdelete(win, NONE);
144     /* CLOSE ALL FILES */
145     d4close_all();
146     fclose(deblog);
147     return (exitcode);
```

```
148 }
149
150
151 /* CalcSchedule
152 *
153 *
154 */
155 int CalcSchedule(long repid)
156 {
157     int done, different, stat;
158     long recno, eqpid, start, duration, repstart, maxstart;
159     time_t tempt;
160     char * ptr, eqpdesc[26], tmp[26];
161     LLLL * replst = NULL, * rlptr = NULL;
162     CONTEXT def;
163     extern HWND win;
164
165     context_save(& def);
166
167     if (BuildList(repid, & replst)) u4error(0, "BUILDLIST ERROR",(char *) 0);
168     fprintf(deblog, "  EQP RECNO List for REPID %ld\n", repid);
169     DumpLLLLNodes(replst, deblog, "    ");
170
171     d4select(eqpreq_dbf);
172     i4select(eqpreq_ndx1);
173
174     repstart = defrepstart;            /* CURRENT TIME + 30 MINUTES                    */
175
176     done = 0;
177     while (! done)
178     {
179         rlptr = replst;
180         while (rlptr)
181         {
182             x4go(rlptr->val);
183             recno = rlptr->val;
184             fprintf(deblog, "    EQP RECNO %ld processing\n", recno);
185             strcpy(eqpdesc, f4str(f4ref("EQPDESC")));
186             c4trim_n(eqpdesc, 25);
187             strupr(eqpdesc);
188             duration = f4long(f4ref("DURATION"));
189             fprintf(deblog, "    Looking for time slot for %s >= %s", eqpdesc, ctime(& repstart
));
190             start = FindEarliest(eqpdesc, duration, repstart, recno);
191             fprintf(deblog, "    Found starttime for %s of %s\n", eqpdesc, ctime(& start));
192             f4r_long(f4ref("START"), start);
193             rlptr = rlptr->next;
194         }
195
196         fprintf(deblog, "    CHECKING IF ALL SAME\n");
197         rlptr = replst;
198         x4go(rlptr->val);
199         maxstart = f4long(f4ref("START"));
200         different = 0;
201         rlptr = rlptr->next;
202         while (rlptr)
203         {
204             x4go(rlptr->val);
205             start = f4long(f4ref("START"));
206             if (start != maxstart)
207             {
208                 maxstart = (start > maxstart ? start : maxstart);
209                 different = 1;
210             }
211             rlptr = rlptr->next;
212         }
213         repstart = maxstart;
214         if (! different)
215         {
216             done = 1;
217             fprintf(deblog, "    POSTING - all same at %s\n", ctime(& repstart));
218             Post(repid, 1);            /* CHANGE ALL STATUS 'T' TO 'A' FOR REPID       */
219         }
220         else
```

```
221         {
222             fprintf(deblog, "    ROLLING BACK - not same, max= %s\n", ctime(& repstart));
223             Post(repid, 0);             /* CHANGE ALL STATUS 'T' BACK TO 'N' FOR REPID   */
224         }
225     }
226     if (replst) FreeLLLL(replst);
227     replst = NULL;
228     rlptr = NULL;
229     context_restore(& def);
230     return 0;
231 }
232
233
234 /* Post
235 *
236 *
237 */
238 int Post(long repid, int mode)
239 {
240     long tmprepid, statusfld;
241     char tmpstatus;
242     int stat;
243     CONTEXT def;
244     extern HWND win;
245
246     context_save(& def);
247
248     d4select(eqpreq_dbf);
249     i4select(eqpreq_ndx1);
250
251     stat = d4seek_double(repid);
252     stat = ((stat == 0) ? 1 : 0);
253     if (! stat)
254     {
255         context_restore(& def);
256         return (-2);                    /* error                                    */
257     }
258     statusfld = f4ref("STATUS");
259
260     if (mode == 1)
261     {
262         while (stat)
263         {
264             tmpstatus = f4char(statusfld);
265             if (tmpstatus == 'T') f4r_char(statusfld, 'A');
266             d4skip(1L);
267             tmprepid = f4long(f4ref("REPID"));
268             if (tmprepid == repid && ! d4eof()) stat = 1;
269             else stat = 0;
270         }
271     }
272
273     if (mode == 0)
274     {
275         while (stat)
276         {
277             tmpstatus = f4char(statusfld);
278             if (tmpstatus == 'T') f4r_char(statusfld, 'N');
279             d4skip(1L);
280             tmprepid = f4long(f4ref("REPID"));
281             if (tmprepid == repid && ! d4eof()) stat = 1;
282             else stat = 0;
283         }
284     }
285     context_restore(& def);
286     return (0);
287 }
288
289
290 /* BuildList
291 *
292 *
293 */
294 int BuildList(long repid, LLLL **ptr)
```

```
295 {
296     long eqprecno, tmprepid;
297     int stat;
298     CONTEXT def;
299
300     context_save(& def);
301
302     * ptr = CreateLLLLNode();
303     d4select(eqpreq_dbf);
304     i4select(eqpreq_ndx1);
305     stat = x4seek_double(repid);
306     stat = ((stat == 1 || stat == 0) ? 1 : 0);
307     if (! stat)
308     {
309         context_restore(& def);
310         return (-2);                    /* error: no equip found for repid in eqpreq     */
311     }
312     while (stat)
313     {
314         eqprecno = d4recno();
315         /* ADD TO LIST */
316         InLLLL(* ptr, eqprecno);
317         x4skip(1L);
318         tmprepid = f4long(f4ref("REPID"));
319         if (tmprepid == repid) stat = 1;
320         else stat = 0;
321     }
322     context_restore(& def);
323     return (0);
324 }
325
326 /* FindEarliest
327 *
328 *
329 */
330 long FindEarliest(char * eqpdesc, long duration, long repstart, long recno)
331 {
332     int result, done, pieceseqp, found;
333     char status, * tmpstr;
334     CONTEXT def;
335     long tmp, tmpstart, tmpduration, tmpend, tmpeqpid, eqpid;
336     long statusfld, startfld, durationfld, eqpdescfld, eqpidfld;
337     LLLL * idlst = NULL, * straddle = NULL, * lookback = NULL, * ptr = NULL;
338     extern HWND win;
339
340     context_save(& def);
341
342     strupr(eqpdesc);
343     c4trim_n(eqpdesc, 25);
344
345     /**** GET ID's FOR EACH EQPDESC IN EQPSUP FILE ****/
346     d4select(eqpsup_dbf);
347     i4select(eqpsup_ndx1);              /* eqpdesc                                       */
348     eqpdescfld = f4ref("EQPDESC");
349     eqpidfld = f4ref("EQPID");
350     result = d4seek_str(strupr(eqpdesc));
351     if (result != 0 && result != 1) return (1);
352     idlst = CreateLLLLNode();
353     InLLLL(idlst, f4long(eqpidfld));
354     x4skip(1L);
355     tmpstr = strupr(f4str(eqpdescfld));
356     c4trim_n(tmpstr, 255);
357     while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
358     {
359         InLLLL(idlst, f4long(eqpidfld));
360         x4skip(1L);
361         tmpstr = strupr(f4str(eqpdescfld));
362         c4trim_n(tmpstr, 255);
363     }
364     pieceseqp = CountLLLLNodes(idlst);
365     fprintf(deblog, "      Found %d instances of %s in eqpsup file\n", pieceseqp, eqpdesc);
366     fprintf(deblog, "      EQP ID List\n");
367     DumpLLLLNodes(idlst, deblog, "      ");
368     /*************************************************************************/
```

```
369
370     /**** LOOK FOR ANY EQPDESC ID NOT USED AFTER REPSTART ****/
371     d4select(eqpreq_dbf);
372     i4select(eqpreq_ndx2);              /* eqpdesc+start                    */
373     statusfld = f4ref("STATUS");
374     startfld = f4ref("START");
375     durationfld = f4ref("DURATION");
376     eqpdescfld = f4ref("EQPDESC");
377     eqpidfld = f4ref("EQPID");
378     result = d4seek_str(strupr(eqpdesc));
379     if (result != 0 && result != 1) return (-1L);
380     tmpstr = strupr(f4str(eqpdescfld));
381     c4trim_n(tmpstr, 255);
382     if (strcmp(tmpstr, eqpdesc) != 0) return (-2L);
383     ptr = idlst;
384     do
385     {
386         found = 0;
387         result = d4seek_str(strupr(eqpdesc));
388         tmpstr = strupr(f4str(eqpdescfld));
389         c4trim_n(tmpstr, 255);
390         while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
391         {
392             tmpeqpid = f4long(eqpidfld);
393             status = f4char(statusfld);
394             tmpstart = f4long(startfld);
395             tmpduration = f4long(durationfld);
396             if ((ptr->val == tmpeqpid) &&
397                     (status == 'A' || status == 'L' || status == 'T') &&
398                     (tmpstart + tmpduration > repstart)
399                 )
400             {
401                 found = 1;
402                 break;
403             }
404             d4skip(1L);
405             tmpstr = strupr(f4str(eqpdescfld));
406             c4trim_n(tmpstr, 255);
407         }
408         if (! found)
409         {
410             fprintf(deblog, "      EQPID %ld COMPLETELY FREE AFTER REPSTART\n", ptr->val);
411             d4go(recno);
412             f4r_char(f4ref("STATUS"), 'T');
413             f4r_long(f4ref("START"), repstart);
414             f4r_long(f4ref("EQPID"), ptr->val);
415             FreeLLLL(idlst);
416             context_restore(& def);
417             return (repstart);
418         }
419         ptr = ptr->next;
420     }
420     while (ptr);
421     /*****************************************************************************/
422
423     /**** ADD ANY ALLOCATED ID's THAT STRADDLE REPSTART TO lookback ****/
424     ptr = idlst;
425     lookback = CreateLLLLNode();
426     d4select(eqpreq_dbf);
427     i4select(eqpreq_ndx2);              /* eqpdesc+...                      */
428     result = d4seek_str(strupr(eqpdesc));
429     if (result != 0 && result != 1) return (-1L);
430     tmpstr = strupr(f4str(eqpdescfld));
431     c4trim_n(tmpstr, 255);
432     if (strcmp(tmpstr, eqpdesc) != 0) return (-2L);
433     do
434     {
435         result = d4seek_str(strupr(eqpdesc));
436         tmpstr = strupr(f4str(eqpdescfld));
437         c4trim_n(tmpstr, 255);
438         while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
439         {
440             tmpeqpid = f4long(eqpidfld);
441             status = f4char(statusfld);
```

```
442             tmpstart = f4long(startfld);
443             tmpduration = f4long(durationfld);
444             if ((ptr->val == tmpeqpid) &&
445                     (status == 'A' || status == 'L' || status == 'T') &&
446                     (tmpstart + tmpduration > repstart) &&
447                     (tmpstart <= repstart)
448                 )
449             {
450                 InLLLL(lookback, ptr->val);
451                 break;
452             }
453             d4skip(1L);
454             tmpstr = strupr(f4str(eqpdescfld));
455             c4trim_n(tmpstr, 255);
456         }
457         ptr = ptr->next;
458     }
458     while (ptr);
459 //      fprintf(deblog,"Lookback List\n");
460 //      DumpLLLLNodes(lookback);
461     /*************************************************************************/
462
463     /**** SEARCH BACKWARD FROM FIRST ID_START>REPSTART TO REPSTART  **/
464     /**** DO NOT SEARCH BACKWARDS IF ID IS IN lookback      ****/
465     d4select(eqpreq_dbf);
466     i4select(eqpreq_ndx2);              /* eqpdesc+start - important                    */
467     result = d4seek_str(strupr(eqpdesc));
468     if (result != 0 && result != 1) return (-1L);
469     tmpstr = strupr(f4str(eqpdescfld));
470     c4trim_n(tmpstr, 255);
471     if (strcmp(tmpstr, eqpdesc) != 0) return (-2L);
472     while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())/* FOR ALL EQPID's              */
473     {
474         tmpeqpid = f4long(eqpidfld);
475         status = f4char(statusfld);
476         tmpstart = f4long(startfld);
477         tmpduration = f4long(durationfld);
478         if ((status == 'A' || status == 'L' || status == 'T') &&
479                 (tmpstart >= repstart)
480             )
481         {
482             if (! InLLLL(lookback, tmpeqpid) && (tmpstart - repstart) >= duration)
483             {
484                 /** BINGO ***/
485                 fprintf(deblog, "      EQPID %ld FREE AT REPSTART\n", tmpeqpid);
486                 d4go(recno);
487                 f4r_char(f4ref("STATUS"), 'T');
488                 f4r_long(f4ref("START"), repstart);
489                 f4r_long(f4ref("EQPID"), tmpeqpid);
490                 FreeLLLL(idlst);
491                 FreeLLLL(lookback);
492                 context_restore(& def);
493                 return (repstart);
494             }
495         }
496         d4skip(1L);
497         tmpstr = strupr(f4str(eqpdescfld));
498         c4trim_n(tmpstr, 255);
499     }
500     /*************************************************************************/
501
502     /**** SEARCH FOWARD FROM EACH ID FOR END > REPSTART **/
503     ptr = idlst;
504     d4select(eqpreq_dbf);
505     i4select(eqpreq_ndx3);              /* eqpdesc+(start+duration)                     */
506     result = d4seek_str(strupr(eqpdesc));
507     if (result != 0 && result != 1) return (-1L);
508     tmpstr = strupr(f4str(eqpdescfld));
509     c4trim_n(tmpstr, 255);
510     if (strcmp(tmpstr, eqpdesc) != 0) return (-2L);
511     while (strcmp(tmpstr, eqpdesc) == 0 && ! d4eof())
512     {
513         tmpeqpid = f4long(eqpidfld);
514         status = f4char(statusfld);
```

```
515         tmpstart = f4long(startfld);
516         tmpduration = f4long(durationfld);
517         if ((status == 'A' || status == 'L' || status == 'T') &&
518                 ((tmpstart + tmpduration) >= repstart)
519             )
520         {
521             if (SlotFoward(d4recno(), duration))
522             {
523                 /** BINGO ***/
524                 tmp = tmpstart + tmpduration;
525                 fprintf(deblog, "      EQPID %ld AVAIL AT %s\n", tmpeqpid, ctime(& tmp));
526                 d4go(recno);
527                 f4r_char(f4ref("STATUS"), 'T');
528                 f4r_long(f4ref("START"), tmpstart + tmpduration);
529                 f4r_long(f4ref("EQPID"), tmpeqpid);
530                 FreeLLLL(idlst);
531                 FreeLLLL(lookback);
532                 context_restore(& def);
533                 return (tmpstart + tmpduration);
534             }
535         }
536         d4skip(1L);
537         tmpstr = strupr(f4str(eqpdescfld));
538         c4trim_n(tmpstr, 255);
539     }
540     /*****************************************************************************/
541
542     context_restore(& def);
543     fprintf(deblog, "ERROR REACHED END OF FindEarliest ROUTINE\n");
544     return (-1L);                       /* SHOULD NEVER REACH THIS EXIT POINT          */
545 }
546
547 int SlotFoward(long rec, long duration)
548 {
549     long statusfld, startfld, durationfld, eqpidfld, eqpdescfld;
550     long curid, curend, tststart, tmpeqpid;
551     char * tmpstr, eqpdesc[255], status;
552     int iseof;
553
554     CONTEXT def;
555
556     context_save(& def);
557
558     d4select(eqpreq_dbf);
559     i4select(eqpreq_ndx2);              /* eqpdesc+start                               */
560     statusfld = f4ref("STATUS");
561     startfld = f4ref("START");
562     durationfld = f4ref("DURATION");
563     eqpdescfld = f4ref("EQPDESC");
564     eqpidfld = f4ref("EQPID");
565
566     d4go(rec);
567     curend = f4long(startfld) + f4long(durationfld);
568     curid = f4long(eqpidfld);
569     status = f4char(statusfld);
570     strcpy(eqpdesc, f4str(eqpdescfld));
571     strupr(eqpdesc);
572     c4trim_n(eqpdesc, 255);
573     do
574     {
575         d4skip(1L);
576         iseof = d4eof();
577         tmpeqpid = f4long(eqpidfld);
578         tmpstr = f4str(eqpdescfld);
579         strupr(tmpstr);
580         c4trim_n(tmpstr, 255);
581         status = f4char(statusfld);
582     }
583     while (((curid != tmpeqpid) ||
584                 (status != 'A' && status != 'L' && status != 'T')) &&
585             ! iseof &&
586             strcmp(eqpdesc, tmpstr) == 0
587         );
588
```

```
589     if ((strcmp(eqpdesc, tmpstr) != 0) || d4eof())
590     {
591         context_restore(& def);
592         return (1);
593     }
594
595
596     if ((f4long(startfld) - curend) >= duration)
597     {
598         context_restore(& def);
599         return (1);
600     }
601     context_restore(& def);
602     return (0);
603 }
```

```
 1 #include <stdio.h>
 2 #include <math.h>
 3 #include <conio.h>
 4 #include <stdlib.h>
 5 #include <dir.h>
 6 #include <time.h>
 7 #include <string.h>
 8 #include <alloc.h>
 9 #include "d4base.h"
10 #include "dw.h"
11 #include "dwmenu.h"
12 #include "dwsystem.h"
13
14 FILE * sched;
15
16 typedef struct
16 {
16     int dbf;
17     int ndx;
18     long rec;
19 }
19     CONTEXT;
20
21 int eqpreq_dbf, eqpreq_ndx;
22
23 int main(int argc, char * argv[])
24 {
25     d4init();
26     sched = fopen("sched", "w");
27     PrtSch();
28     d4init_undo();
29     exit(0);
30 }
31
32 /* PrtSch
33 *
34 *
35 */
36 int PrtSch(void)
37 {
38     int done, i, thisone, r = 0, c = 1;
39     char status, linebuff[300], curdesc[26], * tmpdesc, * timeptr;
40     long start, duration, currepid, minstart, delta = 900L, curtime;
41     long cureqpid, eqpidfld, eqpdescfld, eqpid;
42     long statusfld, startfld, durationfld, repidfld, repid;
43     HWND win;
44     unsigned int key = 0;
45
46
47     /* WINDOW STUFF */
48     win = vcreat(50, 80, ERR, YES);
49     vwind(win, 23, 78, 1, 1);
50     vlocate(win, 1, 1);
51     vframe(win, ERR, FRDOUBLE);
52     vshadow(win, CURRENT, SHADOW75, BOTTOMRIGHT);
53     vmtitle(win, _TOP, CENTERJUST, ERR, " Auto-mode Repair Schedule ");
54
55     eqpreq_dbf = d4use_excl("EQPREQ.DBF");
56     eqpreq_ndx = i4index("tmp", "START", 0, 0);
57
58     statusfld = f4ref("STATUS");
59     startfld = f4ref("START");
60     durationfld = f4ref("DURATION");
61     repidfld = f4ref("REPID");
62     eqpidfld = f4ref("EQPID");
63     eqpdescfld = f4ref("EQPDESC");
64
65     /**** GET EARLIST START ****/
66     minstart = 2147000000;
67     d4top();
68     while (! d4eof())
69     {
70         status = f4char(statusfld);
71         start = f4long(startfld);
```

```
72          if ((status == 'A' || status == 'L' || status == 'C') && start < minstart)
73              minstart = start;
74          d4skip(1L);
75      }
76
77      /* BUILD SCHEDULE */
78      curtime = time(NULL);
79      timeptr = ctime(& curtime);
80      timeptr[strlen(timeptr) - 1] = '\0';
81      vatputf(win, r++, c, "Schedule as of :%s", timeptr);
82      vatputf(win, r++, c, "Each character represents %ld minutes", delta / 60);
83      vatputf(win, r++, c, "Repair Schedule");
84      vatputf(win, r++, c, "%20s|", " ");
85      d4top();
86      while (! d4eof())
87      {
88          for (i = 0; i < 299; i++) linebuff[i] = 32;
89          linebuff[299] = '\0';
90          status = f4char(statusfld);
91          while (! (status == 'A' || status == 'L'))
92          {
93              d4skip(1L);
94              status = f4char(statusfld);
95          }
96          currepid = f4long(repidfld);
97          start = f4long(startfld);
98          duration = f4long(durationfld);
99          Add2Sch(linebuff, minstart, start, duration, delta, status, currepid);
100         TrimSch(linebuff);
101         vatputf(win, r++, c, "REPID %-4ld%10s:%s", currepid, " ", linebuff);
102         while (currepid == f4long(repidfld))
103         {
104             d4skip(1L);
105         }
106     }
107
108
109     eqpreq_ndx = i4index("tmp", "upper(eqpdesc)+str(eqpid,10,0)", 0, 0);
110
111     vatputf(win, r++, c, "Equipment Allocation Schedule");
112     vatputf(win, r++, c, "%20s|", " ");
113     d4top();
114     while (! d4eof())
115     {
116         for (i = 0; i < 299; i++) linebuff[i] = 32;
117         linebuff[299] = '\0';
118         cureqpid = f4long(eqpidfld);
119         strcpy(curdesc, f4str(eqpdescfld));
120         strupr(curdesc);
121         c4trim_n(curdesc, 25);
122         eqpid = f4long(eqpidfld);
123         thisone = 0;
124         while (cureqpid == eqpid && ! d4eof())
125         {
126             status = f4char(statusfld);
127             if ((status == 'A' || status == 'L' || status == 'C'))
128             {
129                 thisone = 1;
130                 start = f4long(startfld);
131                 duration = f4long(durationfld);
132                 tmpdesc = f4str(eqpdescfld);
133                 strupr(tmpdesc);
134                 c4trim_n(tmpdesc, 255);
135                 repid = f4long(f4ref("REPID"));
136                 Add2Sch(linebuff, minstart, start, duration, delta, status, repid);
137             }
138             d4skip(1L);
139             eqpid = f4long(eqpidfld);
140         }
141         TrimSch(linebuff);
142         if (thisone)
143             vatputf(win, r++, c, "%-12s ID %-4ld:%s", curdesc, cureqpid, linebuff);
144     }
145     visible(win, YES, YES);
```

```
146     while (key != ESC)
147     {
148         switch (key = getkey())
149         {
150         case CURUP :
151             vmovedir(win, -1, 0);
152             break;
153         case CURDN :
154             vmovedir(win, 1, 0);
155             break;
156         case CURLF :
157             vmovedir(win, 0, -1);
158             break;
159         case CURRT :
160             vmovedir(win, 0, 1);
161             break;
162         case PGUP :
163             vmovedir(win, -20, 0);
164             break;
165         case PGDN :
166             vmovedir(win, 20, 0);
167             break;
168         default :
169             break;
170         }
171     }
172     clrscr();
173     vexit(0);
174 }
175
176 int context_save(CONTEXT * def)
177 {
178     def->rec = d4recno();
179     def->ndx = i4select(-1);
180     def->dbf = d4select(-1);
181     return (1);
182 }
183
184 int context_restore(CONTEXT * def)
185 {
186     d4select(def->dbf);
187     i4select(def->ndx);
188     if (d4eof()) d4skip(-1L);
189     d4go(def->rec);
190     return (1);
191 }
192
193 int TrimSch(char * linebuff)
194 {
195     int i;
196     for (i = 298; i > -1; i--)
197     {
198         if (linebuff[i] != 32 || i == 0)
199         {
200             if (i) linebuff[i + 1] = '\0';
201             else linebuff[0] = '\0';
202             break;
203         }
204     }
205 }
206
207 int Add2Sch(char * linebuff, long minstart, long start, long duration,
208         long delta, char marker, long repid)
209 {
210     double dbeg, dend;
211     int beg, end, i, buffstart, buffend;
212
213     dbeg = ((double) (start - minstart)) / ((double) delta);
214     dend = dbeg + ((double) duration / ((double) delta));
215     dbeg = (((ceil(dbeg) - dbeg) < .5) ? ceil(dbeg) : floor(dbeg));
216     dend = (((ceil(dend) - dend) < .5) ? ceil(dend) : floor(dend));
217
218     beg = (int) dbeg;
219     end = (int) dend - 1;
```

```
220
221     buffstart = 0;
222     buffend = 298;
223
224     if (beg < 0 || beg > 298) return (-1);
225     if (end > 298) end = 298;
226     for (i = beg; i <= end; i++) linebuff[i] = (char) (repid + 64);
227     return (1);
228 }
229
230
231
232
```